

Programválasztó webalkalmazás Dokumentáció 2025

Készítette:
Serbán Dezső
Szluca Beáta
Toma Árpád

Tartalomjegyzék

| | |
|--|----|
| Programválasztó webalkalmazás | 1 |
| Tartalomjegyzék | 2 |
| Cél | 7 |
| Felhasznált technológiák | 7 |
| Frontend | 7 |
| Backend | 7 |
| Bevezetés, a téma ismertetése, témaválasztás indoklása, szakmai célkitűzés | 9 |
| Hasonló projektek a piacon | 10 |
| Teszt dokumentáció..... | 12 |
| Teszt napló..... | 13 |
| Felhasználói dokumentáció | 14 |
| A program céljának és lényegesebb funkcióinak összefoglalása | 14 |
| Szükséges hardvereszközök és szoftverek felsorolása | 14 |
| Telepítés és indítás lépéseinek ismertetése | 14 |
| Vizualitás..... | 14 |
| A program részletes bemutatása általános felhasználásra | 15 |
| Kezdőoldal | 15 |
| Regisztráció..... | 15 |
| Bejelentkezés | 16 |
| Események | 17 |
| Esemény lista | 17 |
| Esemény részletes oldal | 19 |
| Személyes menü..... | 21 |
| Feliratkozásaim..... | 21 |

| | |
|---|----|
| Kijelentkezés | 22 |
| A program részletes bemutatása admin oldali felhasználásra..... | 22 |
| Események | 22 |
| Címkék | 23 |
| Felhasználók..... | 24 |
| Esemény-tag kapcsoló | 25 |
| Információkérés lehetőségeinek megadása | 27 |
| Összegzés, záró gondolatok | 27 |
| Funkcionális specifikáció | 28 |
| Frontend | 28 |
| Service-k..... | 28 |
| Base: | 28 |
| Local-storage: | 31 |
| Auth Service: | 31 |
| auth-guard Service: | 32 |
| Komponensek | 32 |
| all-events: | 32 |
| card-Pagination: | 34 |
| datepicker-range-popup..... | 35 |
| detailed-event | 36 |
| events-admin-list | 40 |
| events-subscribed:..... | 42 |
| events-tags-link..... | 44 |
| footer | 46 |
| home..... | 46 |
| login..... | 48 |

| | |
|---|----|
| modal | 49 |
| navbar | 49 |
| registration | 50 |
| tags-list | 51 |
| toast-messages | 52 |
| user-settings | 52 |
| users | 52 |
| Fejlesztési lehetőségek | 53 |
| Backend | 55 |
| Általános működés: | 55 |
| Env. fájlhoz szükséges adatok | 55 |
| Útvonalak | 56 |
| Migration | 57 |
| create_cache_table | 57 |
| create_personal_access_tokens_table | 58 |
| create_users_table | 59 |
| create_events_table | 61 |
| create_tags_table | 62 |
| create_subscriptions_table | 63 |
| create_favorites_table | 64 |
| create_comments_table | 64 |
| create_eventtags_table | 65 |
| admin_to_users_table | 66 |
| create_tag_subscriptions_table | 66 |
| add_deleted_at_to_users_table | 67 |
| Seeder | 67 |

| | |
|------------------------------|----|
| DatabaseSeeder..... | 67 |
| Model..... | 68 |
| User | 68 |
| Event | 69 |
| Subscription | 71 |
| Comment..... | 72 |
| Resource | 73 |
| User | 73 |
| Event | 73 |
| Tag | 75 |
| Comment..... | 75 |
| Kontrollerek | 76 |
| Controller..... | 76 |
| ResponseController | 76 |
| VerificationController | 78 |
| UserController | 79 |
| EventController..... | 81 |
| TagController..... | 84 |
| SubscriptionController | 86 |
| Request..... | 88 |
| RegisterRequest..... | 88 |
| LoginRequest..... | 90 |
| ModifyUserRequest..... | 91 |
| EventRequest | 92 |
| SubscribeRequest | 94 |
| TagRequest | 95 |

| | |
|------------------------------------|-----|
| Middleware | 96 |
| AdminAuthMiddleware | 96 |
| Mail | 97 |
| EmailVerification | 97 |
| Provider | 98 |
| AppServiceProvider | 98 |
| Adatbázis | 100 |
| cache tábla | 100 |
| cache_locks tábla | 100 |
| comments tábla..... | 100 |
| events tábla..... | 101 |
| eventtags tábla | 102 |
| favorites tábla | 102 |
| migrations tábla | 103 |
| password_reset_tokens tábla | 103 |
| personal_access_tokens tábla | 103 |
| sessions tábla | 104 |
| subscriptions tábla..... | 105 |
| tags tábla | 105 |
| tag_subscriptions tábla..... | 106 |
| users tábla..... | 106 |

Cél

Ez egy olyan webalkalmazás, ahol a regisztrált felhasználók tudnak keresni különböző szabadidős programokat, és az általuk szimpatikusnak tűnő programokra fel tudnak iratkozni.

A Vizsgarmekünk backend-jét Laravel php keretrendszer, az adatbázist MariaDB server biztosítja. Frontendet egy webes alkalmazás biztosítja, és ezen a felületen egyrészt a felhasználók megkereshetik a számukra legnagyyszerűbb szabadidős programot, amelyre jelentkezhetnek, másrészt az admin jogosultságú felhasználó programokat tehet fel az oldalra, tegeket hozhat létre az eseményekhez, melyekkel jelzi, hogy milyen típusú az adott esemény.

Felhasznált technológiák

Frontend

- angular 18.2.14
 - Nagyon sok könyvtárat tartalmaz.
 - TypeScript alapú
 - Moduláris felépítésű, amely biztosítja hogy a fejlesztők külön-külön is tudjanak dolgozni a különböző modulokon.
 - Kétirányú adatkötések biztosítása.
 - A command line-ban tudunk utasításokat adni az angulárnak, amellyel gyorsan létre tudunk hozni projekteket.
 - Hivatalos támogatása nagyon jó és gyors, dokumentációja megfelelő.
- VS Code 1.98.2
 - Gyors és könnyű használni.
 - Rengeteg bővítményt tartalmaz.
 - Nem igényel sok erőforrást.
 - Van beépített terminálja.
 - Git barát
 - Intellisense / autocomplete sokat segít a programozásnál.
 - Platformfüggetlen.
- Node js 20.18.0
- Bootstrap

Backend

XAMPP Control Panel v.3.3.0

-Adatbázis kezelése a mysql klienssel.

Visual Studio Code

Laravel Framework 11.37.0l

Composer version 2.8.4 2024-12-11 11:57:47

PHP version 8.2.12 (E:\xampp\php\php.exe)

Bevezetés, a téma ismertetése, témaválasztás indoklása, szakmai célkitűzés

A projekt ötlete egy teljesen hétköznapi, ugyanakkor nagyon is valós problémából született meg, amely úgy véljük, a szabadidős tevékenységeket, mozgást és kikapcsolódást kedvelő emberek életében rendre előfordul. Az alapgondolat abból a személyes tapasztalatból indult, hogy, nehézséget jelentett a megfelelő programok, események, túra lehetőségek felkutatása, amelyek az adott időszakban, a lakóhelyünk környezetében éppen elérhetők voltak. Az online keresés ugyan első ránézésre kézenfekvő megoldásnak tűnt, azonban a gyakorlatban ez számos kihívással és buktatóval szembesített.

Azt tapasztaltuk, hogy a programajánló oldalak gyakran tematikusan elkülönülnek egymástól, sokszor egy-egy szűk érdeklődési körre vagy rendezvénytípusra korlátozódnak — legyen szó akár sportról, kulturális programokról, időszakos eseményekről vagy családi rendezvényekről. Tovább nehezítette a helyzetet, hogy ezek az oldalak gyakran pontatlan, elavult és már régóta nem frissített tartalommal működtek. Nem volt ritka, hogy a keresett program már lezajlott, vagy éppen évekkal korábbi esemény adatlapjába futott az érdeklődő, amely nem szolgált aktuális és használható információval.

A projekt célja tehát az volt, hogy egy olyan központi, megbízható és felhasználóbarát felületet hozzunk létre, amely egyszerre több típusú szabadidős tevékenységet és programlehetőséget fog össze, átlátható és szűrhető formában. A rendszer egyik legfontosabb alapelve a naprakészség, azaz a folyamatos frissítés, hogy a felhasználók garantáltan valódi, aktuális ajánlatok közül válogathassanak. A platform lehetőséget biztosít arra, hogy a látogatók személyes érdeklődési körüknek megfelelően böngésszenek a programok között, akár szabadszavas kereséssel, akár különféle címkék alapján.

Ezzel az oldal nem csupán egy információs felületet kínál, hanem egy digitális eszközt is a szabadidő hatékony és élményekben gazdag megtöltéséhez. A projekt során figyelembe kívántuk venni, hogy a mai felhasználók elvárják a gyorsan hozzáférhető, egyszerűen használható és pontos információkat, így a keresési és szűrési funkciókat úgy terveztük meg, hogy minimális időráfordítással elérhetők legyenek a releváns programok. Emellett kiemelt figyelmet fordítottunk a felhasználói

élményre is, hogy az oldal böngészése kényelmes legyen, függetlenül attól, hogy valaki kirándulást, sportrendezvényt, fesztivált vagy éppen egy hétvégi családi programot keres.

A jövőre vonatkozóan úgy véljük, hogy az oldal nemcsak egyéni igényekre reagál, hanem egy általános társadalmi hiányosságot is pótolhat: összehozhatja azokat az embereket, akik aktívan szeretnék eltölteni szabadidejüket. A projekt megvalósítása tehát nem csupán technikai feladat számunkra, hanem egy felhasználói problémára adott átgondolt, átfogó és praktikus válasz.

Hasonló projektek a piacon

A projekt előkészítése során kiemelten fontosnak tartottuk a piacon már elérhető, hasonló szolgáltatások alapos feltérképezését, hogy pontos képet kapjunk az ötletünk piaci relevanciájáról és a meglévő megoldások előnyeiről, illetve korlátairól. A vizsgálat során három olyan platformot elemeztünk részletesen, amelyek felhasználói igények szempontjából részben hasonló célcsoportot szólítanak meg: az Eventland, a Hollesz és a Természetjáró.

A **Hollesz.hu** egy magyar fejlesztésű, eseménykereső platform, amely jól strukturált felületen kínál különböző programokat és rendezvényeket, valamint lehetőséget biztosít szűrésre is. Ugyanakkor a szolgáltatás földrajzi értelemben szűk keresztmetszetet jelent, hiszen esemény kínálata alapvetően Budapest városára és közvetlen környezetére korlátozódik, ami jelentősen csökkenti az országos lefedettséget.

Az **Eventland.eu** oldal a Hollesz-nél szélesebb spektrumot céloz meg. Nem csak eseményeket, hanem programhelyszíneket, éttermeket, kulturális és szórakozási lehetőségeket is bemutat, így összetettebb felhasználói élményt kínál. Ennek ellenére a tartalom náluk is döntően Budapest központú, ezért az ország más régióiból érkező felhasználók számára kevésbé releváns.

A **Természetjáró.hu** ezzel szemben egy specifikusabb célcsoportnak, a természetjárók és kirándulók közösségének kínál jól strukturált túra- és útvonalajánlatokat, valamint kapcsolódó információkat. A platform szakmai tartalma részletes és alapos, azonban tematikus programokra fókuszál, így pl városi, kulturális események nem kapnak helyet.

A versenytárs-elemzés során egyértelműen láthatóvá vált, hogy bár mindhárom vizsgált oldal stabil alapokon működik, egy adott területre —

földrajzilag vagy tematikusan — erősen specializáltak. Ezzel szemben a mi projektünk célja egy sokkal szélesebb spektrumú, országos lefedettségű, tematikailag szabadabb felület megvalósítása volt, amely az aktuális, naprakész tartalmat helyezi középpontba.

Tesztdokumentáció

Egy fejlesztési projekt során, ahol a rendszer különböző komponensei eltérő fejlesztők vagy csapatok kezében vannak, kiemelten fontos szerepet kap a strukturált és folyamatos tesztelési folyamat. Az ilyen jellegű munkamegosztás számos kihívást rejt magában: az eltérő munkastílusok, a párhuzamos fejlesztések, valamint a komponensek közötti függőségek mind olyan tényezők, amelyek növelik a hibalehetőséget.

Éppen ezért már a legkisebb egységek implementálása után azonnali, célzott teszteléssel ellenőriztük az adott funkciók helyes működését. Hibák esetén a javítások gyors és hatékony elvégzésére törekedtünk. Az egységek szintjén nagy hangsúlyt fektettünk arra, hogy az egyes komponensek milyen hatással vannak a környezetükre, azaz az integrációs szintű tesztelés is már korán megkezdődött. Ezt követően komponens szinten (például teljes oldal vagy alrendszer) is rendszeres ellenőrzéseket végeztünk, hiszen egy kisebb változtatás is jelentős hatással lehet a felhasználói felületre vagy a háttérrendszer működésére.

A munkafolyamat szerves része volt a folyamatos csapaton belüli egyeztetés, amely segítette a gyors hibafelismerést és a közös problémamegoldást. Az együttműködés lehetővé tette, hogy hatékonyan kezeljük a nehezebben feltárható, összetettebb hibákat is. A fejlesztési ciklus végén egy átfogó, minden funkciót lefedő tesztelést végeztünk, amely során előre definiált, dokumentált tesztesetek alapján, pontról pontra haladva validáltuk a teljes rendszer működését.

A tesztesetekre alapozott dokumentáció célja az volt, hogy átlátható, követhető és reprodukálható módon rögzítsük az egyes funkciók ellenőrzésének lépéseit, az elvárt eredményeket, illetve az esetleges eltéréseket is. Ez a fajta dokumentáció nem csupán a fejlesztési ciklus lezárása előtt volt hasznos, hanem a későbbi karbantartási, továbbfejlesztési munkák során is támpontot ad.

A részletes naplózásnak és a visszajelzések strukturált rögzítésének köszönhetően a hibák forrása könnyen visszakövethető, a tesztesetek bővíthetők, újrafuttathatók, így a fejlesztési folyamat folyamatosan tanul a korábbi tapasztalatokból. A dokumentációban az is pontosan rögzítésre került, hogy egy-egy funkció milyen rendszerüzeneteket ad vissza sikeres vagy sikertelen végrehajtás esetén, ezzel támogatva a felhasználói élmény egységességét és az esetleges jövőbeli hibaelhárítás hatékonyságát. Ezáltal egy olyan stabil alap jött létre, amely nemcsak a jelenlegi projekt zárását segíti, hanem a rendszer hosszú távú fenntartását és skálázhatóságát is biztosítja.

Tesztnapló



Tesztnaplo.xlsx



Tesztnaplo.zip

Felhasználói dokumentáció

A program céljának és lényegesebb funkcióinak összefoglalása

A programválasztó egy online felület, amelyen különböző területek programlehetőségei, eseményei kerülnek összegyűjtésre. A programok listája folyamatosan frissül, ezáltal a felhasználók bármikor találkozhatnak újdonságokkal, amikor az oldalra látogatnak.

A még nem regisztrált érdeklődők a kezdőoldalon ízelítőt láthatnak az oldalon megtalálható programok közül.

A weboldalon a regisztrált felhasználók böngészhetnek a programlehetőségek között és megtekinthetik azok részleteit. Lehetőségük van továbbá feliratkozni az egyes programokra, majd privát "naplóbejegyzést" írni átélt élményeikről.

Az eseményeket a felhasználók saját igényeik szerint ABC vagy dátum szerint csökkenő, illetve növekvő sorrendbe állíthatják, de opcionálisan kereséssel is szűkíthetik a listázott tételeket.

Szükséges hardvereszközök és szoftverek felsorolása

Az oldal webes felületen érhető el, így bármely eszközön használható, amely rendelkezik aktív internetkapcsolattal. A weboldal bármilyen böngészőn keresztül megnyitható, azonban a leginkább ajánlott programok a Google Chrome, illetve a Mozilla Firefox.

Telepítés és indítás lépéseinek ismertetése

A felület egy webes alkalmazás, ezért nem igényel telepítést és egyéb technológiai felkészülést, előkészületeket. Használatához csupán asztali, vagy mobil eszközre, böngészőre (az ajánlott programok a legjobbak erre a célra), továbbá internetkapcsolatra van szükség.

Vizualitás

A programunk felületének fejlesztésénél a Bootstrap keretrendszer használata kiemelkedő szerepet játszott, mivel jelentősen megkönnyítette a reszponzív és esztétikus design kialakítását. A Bootstrap jól megtervezett grid rendszere lehetővé tette, hogy az oldal különböző képernyőméreteken is optimálisan jelenjen meg, ezzel hozzájárulva a felhasználói élmény javításához. Ez különösen fontos volt a programválasztó programunk esetében, ahol a felhasználók különböző eszközökről kereshetik és böngészhetik a programlehetőségeket. A keretrendszer által biztosított előre megírt stílusok és komponensek — mint például gombok, navigációs sávok és kártyák — lehetővé tették,

hogy gyorsan és hatékonyan tudjuk felépíteni az oldalt, csökkentve ezzel a fejlesztési időt. Ezen túlmenően a Bootstrap hozzáférhetősége és széles körű dokumentációja segített a csapatunknak abban, hogy könnyen implementálhassunk új funkciókat és javítsuk az oldal teljesítményét. Az összességében alkalmazott Bootstrap megoldásaink lehetővé tették, hogy a felhasználók zökkenőmentesen tapasztalják meg a programok böngészését és kezelését bármely eszközön, így fokozva a weboldal vonzerejét és használhatóságát.

A program részletes bemutatása általános felhasználásra

Az ismételtes elkerülése végett a programot aszerint ismertetjük, amely a megértését a leginkább szolgálja. Éppen ezért lehetséges, hogy az adott funkciót, modult nem menüpont alá sorolva részletezzük, hanem onnan kiemelve, külön egységként kezeljük.

Kezdőoldal

Az alkalmazás elsődlegesen a regisztrált, bejelentkezett felhasználók számára kínál funkciókat, a kezdőoldal azonban mindenki számára azonos tartalommal tekinthető meg.

Ezen az oldalon a látogatók néhány, az oldalon megtalálható eseményről, az oldal céljáról, és a regisztráció és bejelentkezés után elérhető funkciókról kapnak információt ízelítőül.

Regisztráció

Amennyiben a látogató még nem rendelkezik érvényes regisztrációval, a kezdőoldalon a Bejelentkezés gombra kattintva, majd a megnyíló felületen a Regisztráció lehetőséget kiválasztva érhető el a regisztrációs form.

Az adatlapon az alábbi adatokat kötelező megadni:

- felhasználónév: minimum 3 karakter, egyedinek kell lennie (még nem szerepelhet az adatbázisban)
- email cím: email formátum kötelező ("@" karakter ellenőrzése), érvényes email címnek és egyedinek kell lennie (még nem szerepelhet az adatbázisban)
- jelszó: minimum 8 karakter, tartalmaznia kell kisbetűt, nagybetűt és számot
- jelszó megerősítése: a két jelszónak egyeznie kell

A regisztrációs gomb abban az esetben aktiválódik, amennyiben a látogató az összes szükséges mezőt kitöltötte.

Abban az esetben, ha bármely beírt tartalom nem felel meg a követelményeknek, arról a felhasználó validációs üzenetként tájékoztatást kap, így minden szükséges támogatás rendelkezésére áll ahhoz, hogy sikeres regisztrációt hajtson végre.

REGISZTRÁCIÓ

Felhasználónév

A név legyen minimum 3 karakter hosszú

Email cím

Nem megfelelő formátum

Jelszó

A jelszó legyen minimum 8 karakter hosszú
A jelszónak tartalmaznia kell kisbetűt, nagybetűt és számot

Jelszó megerősítése

A jelszó legyen minimum 8 karakter hosszú
A jelszónak tartalmaznia kell kisbetűt, nagybetűt és számot

Regisztráció

REGISZTRÁCIÓ

Felhasználónév

Email cím

Jelszó

A jelszónak tartalmaznia kell kisbetűt, nagybetűt és számot

Jelszó megerősítése

A jelszónak tartalmaznia kell kisbetűt, nagybetűt és számot

Regisztráció

A sikeres, vagy valamilyen okból kifolyólag sikertelen regisztrációt a képernyő alján, jobb oldalon felugró kis üzenet jelzi a felhasználó számára.

Sikeres regisztráció!
✕

Hiba történt!
✕

Bejelentkezés

Sikeres regisztrációt követően a felhasználónak lehetősége van a Bejelentkezés gombra kattintva, a belépési adatait megadva bejelentkezni a felületre.

Kötelezően megadandó mezők:

- felhasználónév
- jelszó

Amennyiben valamely kötelező adat nincs megadva, a rendszer validációs visszajelzést küld a felhasználó számára.

BEJELENTKEZÉS

Felhasználónév

tesztelek

Név kötelező

Jelszó

Password12

Jelszó kötelező

Bejelentkezés

Regisztráció

Elfelejtett jelszó

Sikeres belépést követően a felhasználót a kezdőoldalra irányítjuk. Amennyiben valamilyen okból kifolyólag a belépés sikertelen, azt a képernyő alján, jobb oldalon felugró kis üzenet jelzi a felhasználó számára. A hibaüzenetek a következők lehetnek:

- A megadott felhasználónévvel regisztráció nem található
- A felhasználónév vagy jelszó nem megfelelő
- Hálózati vagy szerverhiba történt

Amennyiben a felhasználó rendelkezésére áll egy adott oldalra mutató URL, azonban nincs bejelentkezve a felületre, a jogosultságának ellenőrzése céljából a bejelentkezési felületre irányítjuk.

Sikertelen bejelentkezés esetén a sikertelenség okáról a képernyő alján, jobb oldalon felugró kis üzenetben tájékoztatjuk a felhasználót

Sikeres bejelentkezés után, amennyiben van jogosultsága hozzá, visszairányítjuk arra az oldalra, amelyet a bejelentkezés előtt meg kívánt tekinteni.

Események

Esemény lista

Az oldalon megtekinthető események esemény listák formájában jelennek meg a weboldalon, rácsos elrendezésben.

A listák az esemény kártyákból, a szűrőpanelből, a rendezőből (ABC szerint növekvő, csökkenő, stb.), és az oldal navigációból épülnek fel.

Megjelenítjük továbbá böngészhető események darabszámát is. Oldalanként 12 darab esemény kártya tekinthető meg.

Programok


40 találat | Rendezés: ABC szerint növekvő

Címkék

- ☐ Sport
- ☐ Szórakozás
- ☐ Kultúr
- ☐ Fél napos
- ☐ Egész napos
- ☐ Pár órá
- ☐ Egyéni
- ☐ Csoportos
- ☐ Fizetős
- ☐ Ingyenes

Szűrés Kijelölések törlése

Keresés




20. Wekerlei Garázs vásár Fesztivál

2025-04-26 - 2025-04-27

Wekerletelep, XIX. kerület

A körforgás nem áll meg, ahogy a Wekerlei Garázs vásár Fesztivál

+




50 000 lépés a Holdon Teljesítménytúra

2025-05-10 - 2025-05-10

Hódmezővásárhely

Gyere és fedezd fel a Vásárhely környéki természet szépségét

-




Barkóczy Noémi lemezbemutató

2025-04-03 - 2025-04-03

Magyar Zene Háza


Barkóczy Noémi énekes-gitáros-dalszerző a 2010-es évek elején

+




Budapest FotóFesztivál

2025-03-28 - 2025-05-14



Budapest - városi séták

2025-05-14



Debrecen városi nyomozós program - CityFox

Kivételt képez ez alól a Kezdőoldal, amelyen néhány kiemelt esemény kártyája tekinthető meg, a szűrőpanel és egyéb esemény lista elemek nem képezik részét az oldal funkcionalitásának.

Az esemény kártyákon megtekinthetők az események kiemelt adatai, úgymint:

- esemény képe
- esemény megnevezése
- esemény kezdő és záró dátuma (amennyiben nincs dátuma, az esemény "Állandó", tehát bármikor részt lehet venni rajta)
- esemény helyszíne
- részlet az esemény leírásából

A kártyák részét képezi továbbá egy gomb, amely az adott eseményre történő fel- és leiratkozást hivatott kezelni (a funkció bővebb leírását ld: Személyes menü/Feliratkozásaim, illetve Esemény részletes oldal).

Abban az esetben, ha a felhasználó az adott eseményre még nem iratkozott fel, a gombra kattintva feliratkozik, amennyiben már fel van iratkozva, leiratkozik az eseményről. A feliratkozás sikerességéről vagy sikertelenségéről a képernyő alján, jobb oldalon felugró kis üzenetben tájékoztatjuk a felhasználót .

A listákban alapértelmezetten ABC szerint növekvő sorrendben tekinthetők meg az események. A felhasználónak azonban lehetősége van igényei szerint módosítani ezen, és ABC szerint csökkenő, vagy dátum szerint növekvő, illetve csökkenő sorrendbe állítani a kártyákat.

18

A felhasználónak lehetősége van a megjelenített eseményeket szűrni is. Ezt megteheti szabadszavas keresést alkalmazva, vagy az eseményhez kapcsolt címkék közül választva.

Szabadszavas keresés esetén miután a felhasználó a kereső mezőbe beírta a kívánt feltételt, rá kell kattintani a mező végén elhelyezkedő nagyító (keresés) ikonra. Az oldal sikeres találat esetén a feltételek szerinti találati eredményt fogja megmutatni.

A szabadszavas szűrés törléséhez rá kell kattintani a mezőben megjelent x (törlés) ikonra.

Címkék szerinti szűréshez ki kell választani a megfelelő címkékhez kapcsolódó jelölőnégyzeteket, majd rá kell kattintani a Szűrés gombra. Az oldal sikeres találat esetén a feltételek szerinti találati eredményt fogja megmutatni.

A címkés keresés törléséhez rá kell kattintani a Szűrés törlése gombra.

Amennyiben a felhasználó olyan feltételt adna meg, amely nem hoz találati eredményt, erről az alábbi üzenetben tájékoztatjuk.

Nincs találat

Sajnos a megadott feltételek, nem hoztak eredményt. Kérlek, a szabadszavas keresést és a címkékre szűrést ne használt egyszerre.

Esemény részletes oldal

Adott esemény kártyáján a képre kattintva megnyílik annak részletes oldala. Ezen a felületen további információk tekinthetők meg az eseménnyel kapcsolatban, a felhasználó fel- és leiratkozhat az eseményről, illetve az eseményre történő feliratkozás után a felhasználó privát élménybeszámolót is rögzíthet hozzá.

Az oldalon megtekinthető az esemény neve és képe. A kép alatt balra az esemény leírását és egyéb információit tekintheti meg a felhasználó. Alapértelmezetten a Leírás fül tartalma látható, amely az adott eseményhez kapcsolódó teljes leírást megjeleníti. Az Információkat kiválasztva pedig olyan egyéb adatokat jelenítünk meg mint:

- az eseményhez kapcsolt címkék
- az esemény weboldalának címe
- az esemény letölthető gpx fájlja (amennyiben kapcsolódik hozzá, pl egy túra esetén)

Amennyiben a felhasználó feliratkozott az adott eseményre, megjelenik egy harmadik választható fül is, az Élménybeszámoló.

Erre kattintva a felhasználó privát bejegyzést írhat az átélt élményeiről, vagy megtekintheti korábban már feljegyzett gondolatait, mintegy sajátos "napló bejegyzésként".

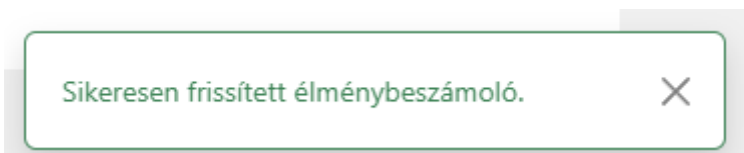
Amennyiben a felhasználó még nem rögzített az eseményhez bejegyzést, az Élménybeszámoló fülre kattintva egy tájékoztató szöveget olvashat a funkcióról.



Állandó esemény esetén az Élmény hozzáadása gomb aktív, a felhasználó bármikor rögzíthet hozzá bejegyzést. Amennyiben kezdő és záró dátummal rendelkezik az esemény, a gomb az esemény kezdetét követő napon válik aktívvá.

Az Élmény hozzáadása gombra kattintva megnyílik a felhasználó számára egy mező, amelyben rögzítheti élményeit. A bejegyzés mentését követően, később lehetősége van annak tartalmát módosítani, a beszámolót törölni azonban nem áll módjában.

A rögzítés, módosítás sikerességéről vagy esetleges sikertelenségéről a képernyő alján, jobb oldalon felugró kis üzenetben tájékoztatjuk a felhasználót.



Élménybeszámoló rögzítése után a felhasználó számára a Leiratkozás gomb inaktívvá válik, ezzel biztosítva számára, hogy még véletlenül se vesszenek el bejegyzései.

A kép alatt jobb oldalon megtekinthető:

- az esemény kezdő és záró dátuma (amennyiben nincs dátuma, az esemény "Állandó", tehát bármikor részt lehet venni rajta)
- az esemény kezdő és záró időpontja (amennyiben van)
- az esemény helyszíne
- az esemény címe
- az esemény státusza
- az esemény néhány kiemelt címkéje

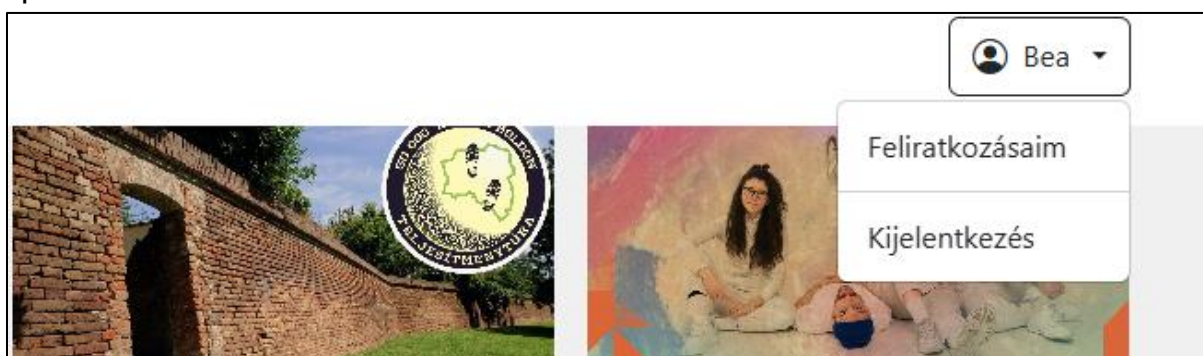
A jobb oldali blokk azt a célt szolgálja, hogy a felhasználó a részletes oldalt megnyitva, a lehető leggyorsabban a legfontosabb információk birtokába jusson.

Az oldal ajlán láthatók a Hasonló események ajánló. Itt olyan eseményeket jelenítünk meg a felhasználó számára, amelyek hasonló címkékkel vannak ellátva, mint az éppen megtekintett esemény, így akár a részletes oldalról tovább lehet böngészni az érdeklődési kör alapján.

Személyes menü

Sikeres bejelentkezés esetén a menüsáv jobb oldalán megjelenik a felhasználó számára egy lenyíló menü, amelyen a saját felhasználónevét láthatja.

Erre rákattintva megjelennek számára azon menüpontok, amelyek specifikusan csak rá vonatkozó információkat és funkciókat tartalmaznak.



Feliratkozásaim

A felhasználónak lehetősége van feliratkozni eseményekre, amelyek ezt követően megtekinthetők a Feliratkozásaim menüpontot kiválasztva.

Ennek előnye, hogy ezen kiemelt események nem vesznek el a több száz esemény között, hanem könnyedén megtalálhatók és átlapozhatók a menüponton keresztül.

Feliratkozást követően adott esemény részletes oldalán megjelenik az Élménybeszámoló fül, amelyet kiválasztva a felhasználó (az esemény kezdő dátumának eltelte után) az Élmény hozzáadása gombra kattintva lejegyezheti átélt tapasztalatait.

Amennyiben a felhasználó nem szeretné itt látni valamely eseményt, akár a listából, akár az esemény részletes oldalán a Feliratkozás gombra kattintva eltávolíthatja azt a Feliratkozásaim oldalról.

Ha adott eseményhez rögzített már a felhasználó élménybeszámolót, a Feliratkozás gomb inaktív, a feliratkozás nem lehetséges.

Kijelentkezés

A személyes menüben megtalálható Kijelentkezés gombra kattintva a felhasználót azonnal kijelentkeztetjük az oldalról. Onnantól kezdve a bejelentkezett funkciókat és információkat nem fogja elérni.

A program részletes bemutatása admin oldali felhasználásra

A program_neve három jogosultsági szintet különböztetünk meg:

- általános felhasználó
- admin
- superadmin

Az általános felhasználók érik el a felhasználói felületet és annak funkcionálisait. Az admin és superadmin joggal rendelkezők érik el az admin felületet és annak funkcionálisait.

Szuperadmin joggal rendelkező felhasználó csak 1 db van, ő az aki teljeskörűen minden funkcióhoz hozzáfér a felületen, amely kiszervezésre került az oldalra.

Események

Ezen az oldalon megtekinthetők az adatbázisba már rögzített események és azok információi:

- id (egyedi azonosító)
- neve
- kezdő és záró dátuma (ha van)
- kezdő és záró időpontja (ha van)
- leírása
- ország
- helyszín megnevezése
- cím
- weboldal
- kép (új felvételénél, vagy meglévő módosításánál az elérési útvonal megadása szükséges)
- GPX fájl (új felvételénél, vagy meglévő módosításánál az elérési útvonal megadása szükséges)

Az eseményekhez lehet újat felvenni a felső, Új program felvétele formot kitöltve. Amennyiben a felhasználó nem megfelelően tölti ki az adatlap mezőit, arról a mezők alatt megjelenő validációs információban tájékoztatjuk.

Új program felvétele

Program neve

Túl rövid esemény név.

Leírás

Kezdő dátum

Érvényes kezdeti dátum szükséges.
Kezdeti dátum formátuma helytelen.
Kezdeti dátum a befejezési dátumnál nem lehet korábban.

Záró dátum

Érvényes befejezési dátum szükséges.
Befejezési dátum formátuma helytelen.
Befejezési dátum a kezdeti dátumnál nem lehet korábban.

Kezdő időpont

Záró időpont

Ország

Weboldal

Program helyszín megnevezése

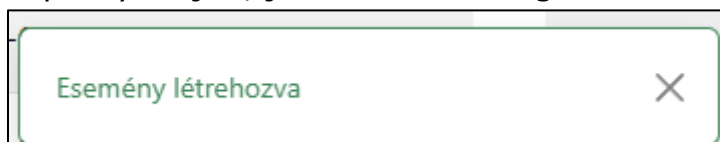
Kép

Program címe

GPX fájl

Mentés

Az adatlap megfelelő kitöltését követően a Mentés gombra kattintva visszaigazolást kap a felhasználó a sikeres, illetve sikertelen rögzítésről, a képernyő alján, jobb oldalon felugró kis üzenetben.



A felhasználónak lehetősége van adott esemény sorában a ceruza (módosítás) ikonra kattintva módosítani az esemény adatait. Módosítás esetén tájékoztatjuk a felhasználót arról, hogy a bevitt adatok megfelelők-e, továbbá a módosítás mentésének sikerességéről, vagy sikertelenségéről. Adott esemény sorában a Törlés gombra kattintva lehetőség van az esemény törlésére. Ennek sikerességéről, vagy sikertelenségéről szintén a megszokott módon tájékoztatjuk a felhasználót.

Címkék

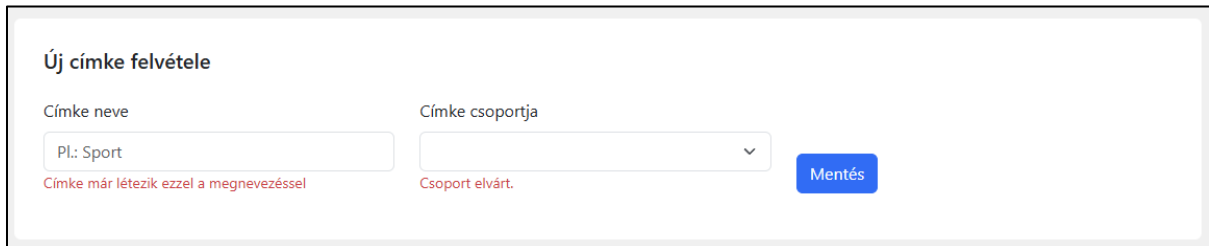
Ezen az oldalon tekinthető meg az adatbázisba rögzített címkék listája, azok kapcsolódó adataival.

Megtekinthető információk:

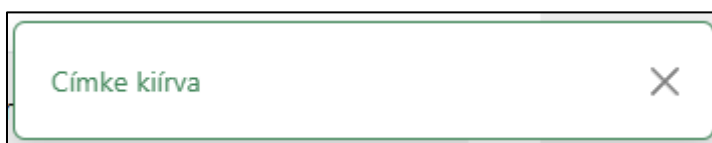
- id (egyedi azonosító)
- neve
- csoportja

Az oldalon továbbá lehetőség van új címke felvételére a felső Új címke felvétele form kitöltésével.

Amennyiben a felhasználó nem megfelelően tölti ki az adatlap mezőit, arról a mezők alatt megjelenő validációs információban tájékoztatjuk.



Az adatlap megfelelő kitöltését követően a Mentés gombra kattintva visszaigazolást kap a felhasználó a sikeres, illetve valamilyen okból történő sikertelen rögzítésről, a képernyő alján, jobb oldalon felugró kis üzenetben.



A felhasználónak lehetősége van adott címke sorában a ceruza (módosítás) ikonra kattintva módosítani annak adatait. Módosítás esetén szintén tájékoztatjuk a felhasználót arról, hogy a bevitt adatok formátuma megfelelő-e, továbbá a módosítás mentésének sikerességéről, vagy sikertelenségéről.

Adott címke sorában a Törlés gombra kattintva lehetőség van a címke törlésére. Ennek sikerességéről, vagy sikertelenségéről szintén a megszokott módon tájékoztatjuk a felhasználót.

Felhasználók

A felhasználók menüpontra belül megtekinthető a program_neve weboldalon regisztrált, aktív státuszú felhasználók listája, illetve a hozzájuk kapcsolódó adatok:

- id (egyedi azonosító)
- név
- email cím
- jogosultsági szint

A táblázatban a ceruza (módosítás) ikonra kattintva módosítani is lehet bizonyos adatokat, ez jogosultsági szintenként eltérő.

Amely felhasználó módosítására nincs jogosultsága a belépett felhasználónak, annak sorában nem jelenik meg a módosítás lehetőségét jelző ikon.

A superadmin jogosultsággal rendelkező felhasználó módosítani tudja az általános felhasználó, illetve az admin felhasználó:

- felhasználónevét
- email címét

- jogosultsági körét: adminról általános felhasználóra, vagy általános felhasználóról adminra

Tehát minden, az oldalon regisztrált felhasználó általános felhasználóként kerül rögzítésre a rendszerbe. Amennyiben admin jogosultságot szeretnének adni az adott felhasználó számára, ezt a superadmin tudja megtenni.

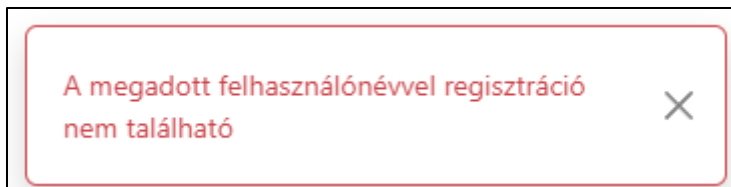
Az admin jogosultsággal rendelkező felhasználók módosítani tudják az általános felhasználók:

- felhasználónevét
- email címét

Az adatlap megfelelő kitöltését követően a Módosítás gombra kattintva visszaigazolást kap a felhasználó a sikeres, illetve valamilyen okból történő sikertelen rögzítésről, a képernyő alján, jobb oldalon felugró kis üzenetben.



A superadmin jogosultsággal rendelkező felhasználónak továbbá lehetősége van adott felhasználót a Törlés gombra kattintva "inaktíválni". Sikeres inaktíválást követően az adott felhasználó a továbbiakban nem fog megjelenni az aktív felhasználók listájában, továbbá nem fog tudni a törölt regisztrációjával belépni a felületre.



A sikeres, illetve valamilyen okból sikertelen törlésről, a képernyő alján, jobb oldalon felugró kis üzenetben tájékoztatjuk a felhasználót.

Esemény-tag kapcsoló

Minden eseményhez érdemes legalább egy címkét kapcsolni az adatbázisban, amit ezen a felületen lehet megtenni. Itt láthatjuk továbbá a már összekapcsolt esemény - címke párokat, illetve azon események listáját is, amelyekhez még egyáltalán nincs esemény kapcsolva.

Az összekapcsolt esemény - címke párokról az alábbi adatok tekinthetők meg a táblázatban:

- esemény azonosító
- esemény neve
- címke azonosító
- címke neve

- címke csoportja

Mind a már összekapcsolt adatokat tartalmazó táblázatban, mind a címke nélküli eseményeket listázó táblázatban lehetőség van szabadszavas keresésre a tábla fölött elhelyezett keresés segítségével.

Az oldalon a felső Új esemény és címke kapcsolat felvétele blokkban van lehetőség új esemény - címke kapcsolatot rögzíteni.

Ehhez a felhasználónak rá kell keresnie vagy a kívánt esemény azonosítójára, vagy az esemény nevére. Esemény azonosítóra keresve csak a teljesen azonos találatot adja vissza a rendszer, az esemény nevére keresve azonban minden lehetséges találatot kilistáz a felhasználó számára.

Amennyiben nincs megfelelő találati eredmény, arról egy megjelenő információs blokkban tájékoztatjuk a felhasználót.

ESEMÉNY ÉS CÍMKE ÖSSZEKAPCSOLÁS

Új esemény és címke kapcsolat felvétele

Keresés ⓘ

Esemény azonosító
Esemény neve

Nincs találat

Lehetséges okok:

- Egyszerre adtad meg az Esemény azonosítóját és az Esemény nevét. Kérlek, csak az egyik adatot add meg.
- A keresett tartalom nem létezik az adatbázisban

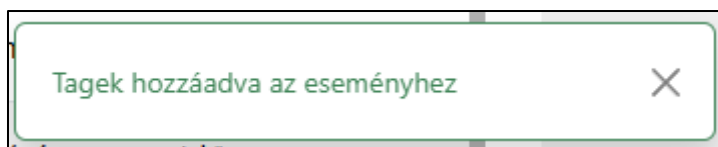
A találati listában megjelenik az esemény azonosítója, neve, illetve a hozzá kapcsolható címkék listája. A címkék listájából többet is ki lehet választani, a rendszer csak azokat a címkéket fogja hozzákapcsolni az adott eseményhez, amely még nem található meg az esemény - címke kapcsolati táblában.

Amennyiben a felhasználó címke kiválasztása nélkül kattint rá a Mentés gombra, a képernyő alján, jobb oldalon felugró kis üzenetben figyelmeztetjük erre.

A címke nincs kiválasztva!

×

A címkék kiválasztását követően a Mentés gombra kattintva visszaigazolást kap a felhasználó a sikeres, illetve valamilyen okból történő sikertelen rögzítésről, a képernyő alján, jobb oldalon felugró kis üzenetben.



Sikeres rögzítést követően az adatok azonnal megjelennek az esemény - címke kapcsolatot listázó táblázatban.

Információkérés lehetőségeinek megadása

Amennyiben információkat szeretne kapni, részleteket elérni a weboldal működésével kapcsolatban, hibás működés tapasztalása esetén, esetleg fejlesztési javaslatai lennének, az alábbi elérhetőségen vehető fel a kapcsolat az oldal fejlesztőivel:

arbedeprojekt@gmail.com

Összegzés, záró gondolatok

A projekt megvalósításának célja az volt, hogy egy olyan könnyen használható felületet hozzunk létre, ahol a felhasználók gyorsan és egyszerűen találhatnak számukra érdekes, szabadidős programokat. Fontos szempontként kezeltük, hogy a keresés és böngészés folyamata ne okozzon felesleges bonyodalmat, hanem gördülékenyen és átlátható módon segítse a döntést. A tervezési szakaszban több ötlet és funkció is felmerült, amelyeket szerettünk volna megvalósítani, de a fejlesztés előrehaladtával kiderült, hogy a rendelkezésre álló idő és kapacitás szűkössége miatt engednünk kell az elvárásainkból. Ezzel együtt világossá vált az is, hogy az előzetes tervezés során pontosabb becslésre és átgondoltabb ütemezésre lett volna szükség.

A fejlesztési folyamat nem volt mentes a kihívásoktól. Időnként nehézséget okozott az információk hiánya, az egymástól függetlenül zajló feladatok összehangolása, valamint a szoros határidők miatti feszültség is próbára tette a csapatot. Ugyanakkor ezek a helyzetek sok tanulságot hoztak, hiszen a közös egyeztetések és a problémák megoldása során nemcsak a szakmai tudásunk, de az együttműködési készségünk is fejlődött. Az elkészült felület stabil kiindulópont lehet a további bővítésekhez és finomításokhoz. A munka során megtapasztaltuk, hogy egy projekt sosem tekinthető teljesen lezártnak, inkább folyamatos fejlesztési lehetőségként kell rá tekinteni, ahol mindig van mód a fejlődésre és jobb megoldások kidolgozására.

Funkcionális specifikáció

Frontend

Service-k

Base:

Feladata a backend és a frontend összekötése.

Változók:

- backendUrl => Ez a backendünk alap elérési útja.
- eventsAllSub=>Ebben tároljuk az összes szabadidős programot, amelyeket a backend küld.
- myEvents=>A bejelentkezett felhasználónak az általa felíratkozott eseményeket tartalmazza.
- tagsSub=>Ebben tárolódnak a tegek, amelyek az események szűrésénél segítenek abban, hogy most az adott esemény milyen (pl:Spar futás az Sport teggel ellátott).
- eventsWithTags=>Itt megkapjuk a tegek és az események közötti kapcsolatot.
- dataUsersSub=>Backendtől megkapja az admin jogosultságú felhasználó, hogy kik vannak regisztrálva a weboldalra.
- myExperiences=>Ebben tárolódik a felhasználónak a felíratkozott eseményekhez hozzáfűzött kommentjei.

Metódusok:

- downloadAllUsers()=>A backend-től lekérjük az összes felhasználót, amely ahhoz kell, ha admin jogú felhasználó van bejelentkezve, akkor műveleteket tudjon végrehajtani a felhasználókon (admin jog adása vagy elvétele).dataUsersSub-ban tárolódik a backend válasz.
- getEventById(id:number)=>Az esemény azonosító számát elküldve a backendnek, válaszként visszakapjuk az eseményt. Ez azért kell, mert ha a felhasználó, ha az eseményhez kapcsolódó képre kattint, akkor a részletes oldalon megkapjuk a teljes eseményt részletes adatokkal és plusz funkciókkal.
- updateUser(data:any)=>Ha az admin jogosultsággal rendelkező felhasználó, már meglévő felhasználónak a jogosultságán szeretne

változtatni. Bemenő paraméter maga az esemény, amelyen szeretnénk változtatni.

- `deleteUser(data:any)=>` Ha az admin jogosultsággal rendelkező felhasználó el akar távolítani egy felhasználót az adatbázisból. Bemenő paraméter maga az esemény, amelyet ki szeretnénk törölni
- `getAllUsers()`=> Visszatér a `dataUsersSub`-bal, ami tartalmazza a weboldalon regisztrált felhasználókat.
- `getAll()`=> A weboldalon tárolt összes eseménnyel tér vissza.
- `downloadAll()`=> Ez megtölti azt a változót az eseményekkel (`eventsAllSub`), amit a `getAll()` metódus hív meg.
- `getAllMyEvents()`=> Ez tölti meg a `myEvents` változót, ami a bejelentkezett felhasználó feliratkozott programjait tartalmazza.
- `subscribeEvent(data:any)=>` A bejelentkezett felhasználó ezzel a metódussal tud feliratkozni az általa tetszetősnek tartott eseményre. A bemenő paraméter maga az esemény, amelyre fel szeretne iratkozni a felhasználó.
- `unsubscribeEvent(data:any)=>` A bejelentkezett felhasználó ezzel a metódussal tud leiratkozni arról az eseményről, amelyre már feliratkozott, de meggondolta magát.
- `newDataWeb(data:any)=>` Admin jogosultsággal rendelkező felhasználó ezáltal tud új eseményt regisztrálni az oldalra. A bemenő paraméter maga az esemény, amelyről le szeretne iratkozni a felhasználó.
- `updateDataWeb(data:any)=>` Admin joggal rendelkező felhasználó ezáltal tud meglévő eseménynél adatot módosítani. A bemenő paraméter maga az esemény, melyet szeretne módosítani az admin jogosultsággal rendelkező felhasználó.
- `deleteDataWeb(data:any)=>` Admin joggal rendelkező felhasználó ezáltal tud meglévő eseményt eltávolítani a weboldalról. A bemenő paraméter maga az esemény, amelyet törölni szeretene az admin jogosultsággal rendelkező felhasználó.
- `downloadAllTags()`=> A `tagsSub` változóba betölti az összes teget, amely alapján tudjuk majd csoportosítani és szűrni az eseményeket.
- `newTagWeb(data:any)=>` Admin jogosultságú felhasználó tud új teget felvenni. A bemenő argumentum az admin jogosultságú által meghatározott új teg adatai.
- `deleteTagWeb(data:any)=>` Admin jogosultságú felhasználó tud meglévő teget eltávolítani a backend adatbázisából. Bemenő argumentum az adott teg, amelynek az id értékét használjuk fel a törölni kívánt teg kiválasztására, majd törlésére.

- `updateTagWeb(data:any)=>` Admin jogosultságú felhasználó tud meglévő teget módosítani. A bemenő paraméter maga a teg, amit az admin joggal rendelkező felhasználó módosítani szeretne
- `searchEvent(data:any)=>` A backend segít szűrni a szabadszavas keresésben. A bemenő paraméter egy string lesz, amit megkap a backend, majd ez alapján válaszul megadja, azokat az eseményeket, amelyek megfelelnek a stringnek.
- `getDateFromDatePicker(date:any)=>` A keresésnél fogjuk használni, mikor dátum szerint szeretnénk keresni, **FEJLESZTÉSI LEHETŐSÉG**
- `getEventsWithTags()`=> Megkapjuk az események és tegek közötti kapcsolatot, amely majd a teg alapú szűrésnél szükséges.
- `attachTagToEvent(data:any)=>` Amikor egy eseményhez egy admin jogosultságú felhasználó egy teget szeretne hozzákapcsolni. A bemenő paraméter tartalmazza az esemény id-ját és a teg id-ját, majd ezt átadja a backendnek, amely összekapcsolja a kettőt, és elmenti az adatbázisba.
- `detachTagFromEvent(data:any)=>` Admin jogú felhasználó le akarja választani az eseményről a korábban hozzáadott teget. A bemenő paraméter tartalmazza az esemény id-ját és a teg id-ját, és ez alapján a backend a meghatározott id-jú teget eltávolítja a meghatározott id-jú eseményről.
- `show(message:string, type:'success' | 'danger' | 'warning' | 'info' = 'success')`=> Backendtől kapott üzenetek megjelenítése, mely szólhat sikeres végrehajtásról vagy akár hibáról.
- `remove(message:string)=>` Üzenet eltávolítása 5 mp elteltével a `show()` metóduson belül.
- `updateUserExperience(data:any)=>` A bejelentkezett felhasználó a feliratkozott eseménynél tudja korábbi kommentjét módosítani.
- `getMyExperience()`=> A bejelentkezett felhasználó a feliratkozott eseményeinél ha van kommentje, az megjelenik.
- `toSort(terms:string, events:any)=>` Ez metódus valamilyen feltétel szerint fogja megjeleníteni az eseményeket a weboldalon (ABC sorrend, Dátum szerinti rendezés). Bemeneti paramétereknél a `terms` a feltétel (növekvő vagy csökkenő), `events` pedig maguk az események, amelyeket valamilyen sorrendbe szeretnénk rendezni.
- `filterByTag(tagIds:any[])`=> A backend felé küldünk teg id-kat és visszakapjuk a hozzátartozó eseményt. Ez a teg alapú szűréshez kell. A bemenő paraméter tartalmazza a tegek id-ját, amelyet egymás után egyesével elküldjük a backend-nek, majd a kapott válasszal dolgozunk tovább.

Local-storage:

Feladata, hogy mikor a felhasználó a weboldalon a lapok között vált, akkor megmaradjon a bejelentkezésnél keletkezett token, jogosultsági szint és felhasználónév.

Metódusok:

- `setItem(key: string, value: string):void=>` Kulcs értékpár szerint elmenti az adatokat.
- `getItem(key:string):string |null =>` Kulcs alapján visszatér a kulcshoz tartozó értékkel.
- `removeItem(key:string):void=>` Kulcs alapján törli az értéket a local-storage-ból.
- `clear():void=>` Törlődik a teljes tartalom a local-storage-ból. A felhasználó kijelentkezése után használjuk.

Auth Service:

Feladata, a bejelentkező felhasználó azonosítása, és sikeres bejelentkezés esetén a user adatainak tárolása.

Változók:

- `userSub=>` A bejelentkező user adatait tárolja.
- `backendUrl =>` Ez a backendünk alap elérési útja.
- `saveBackendMessage=>` A regisztrációs folyamatnál fellépő hibaüzeneteket tárolja.
- `messages =>` Ez egy BehaviorSubject típusú tömb, amely olyan objektumokat tartalmaz, amely 2 mezőből áll: text és type mező melyek string típusúak. Ez fogja tárolni a toast message-nek az üzeneteit.
- `messages$=>` Ez a változó, a messages-t kapja meg observable-ként, és ezt arra használjuk, hogy más komponensek, ebből fogják megkapni az üzeneteket, mivel ezt csak olvasni lehet, módosítani nem.

Metódusok:

- `showToast(message : string, type: 'success' | 'danger' | 'warning' | 'info' = 'success')=>` Backend üzenetek megjelenítése.

- `removeToast(message: string)=>` Üzenet eltávolítása 5 mp után a `showToast()` metódusból.
- `getLoggedInUser()=>` A bejelentkező user adatait megkapjuk a `userSub` változóból. Arra használjuk, mikor az érdekel minket, hogy ki van jelenleg bejelentkezve.
- `setLoggedInUser(user:any)=>` A bejelentkezés folyamatnál a `userSub`-változóba tölti be az adatokat. A `user` paraméter fogja tartalmazni a sikeresen bejelentkezett felhasználó adatait.
- `registrationUserOnlaravel(nameArg: string, emailArg: string, passwordArg: string, confirm_passwordArg: string)=>` Felhasználó regisztrálása a weboldalra.
- `loginWithLaravel(nameArg: string, passwordArg: string)=>` Felhasználó bejelentkeztetése.
- `logoutUserFromLaravel()=>` A bejelentkezett felhasználó kijelentkeztetése, mely során töröljük a tokenjét az adatbázisból és az összes vele kapcsolatos adatot a `local storage`-ból.

auth-guard Service:

Feladata, a mikor valamilyen interakciót akarnak végrehajtani a látogatók a weboldalon és nincsenek még bejelentkezve, akkor átnavigálnak a bejelentkezést végrehajtó komponensre.

Komponensek

all-events:

Feladata, az összes esemény megjelenítése.

Változók:

- `events=`> Ez a változó tartalmazza a backendből betöltött összes eseményt.
- `currentPage=`> Az oldalon belül a az események megjelenítésénél használjuk.
- `itemsPerPage=`> Az oldalon belül a az események megjelenítésénél használjuk. Egy oldalon hány darab eseménykártya lehet.
- `eventStartDateNull=`> Mikor az backend felől érkezett esemény nem tartalmaz kezdő dátumot, akkor az ebben tárolt szöveg jelenik meg a programkártyán.
- `eventEndDateNull=`> Mikor az backend felől érkezett esemény nem tartalmaz befejező dátumot, akkor az ebben tárolt szöveg jelenik meg a programkártyán.

- selectedOption=> Itt tárolódik, hogy a felhasználó ABC sorrendben, vagy dátum szerinti sorrendben szeretné megjeleníteni a programokat.
- tags => Ebben tároljuk a backendtől megszerzett tegeket.
- selectedTags=> A felhasználó által kiválasztott tegeket ebben tároljuk, és ezt használva valósítjuk meg a teg alapú szűrést.
- tagSearch=> Ez egy bool, ami azt mondja meg, hogy éppen keresek e teg alapján vagy sem.
- tagsOfEvents:any[] => Csak azok a tegek vannak ebben a tömbben, amelyhez hozzá vannak kapcsolva események.
- attachedDatas:any[] =>getEventsTags() metódusnál használjuk szerepe, hogy tárolja a teg-esemény kapcsolatot.
- searchControl=> Ebben tárolódik a felhasználó által begépelt szavak, és amely a szabadszavas keresés megvalósításához szükséges.
- isSearch=> Ez egy bool, ami azt mondja meg, hogy éppen használom e a szabadszavas keresést vagy nem.
- commonSearchResults: Ebbe töltjük bele a searchTagResults vagy searchResults eseményeit, de ha egyszerre használom a kettő fajta keresést, akkor a kettőnek uniója töltődik bele ebbe a változóba.
- userEventsWithTags:any[]=> Tárolja
- userEvents => Ebben tárolódnak azok az események, amelyekre a bejelentkezett felhasználó feliratkozott.

Metódusok:

- getDataFromApi()=> Az események betöltése a weboldalra.
- changePage(page:number)=> Megvalósítja a lapozást az oldalon.
- get paginatedEvents():any[]=> Ez a getter segít megvalósítani a lapozáshoz szükséges logikát, amelyet az összes eseményen alkalmazunk.
- get paginatedSearchedEvents():any[]=>Ez a getter segít megvalósítani a lapozáshoz szükséges logikát, amelyet a már leszűrt eseményeken (címkékkel vagy szabad szavasan keresett) alkalmazunk.
- toSort()=> A weboldalon az eseményeknek az ABC- vagy dátum szerinti növekvő vagy csökkenő sorrendbe állítását valósítja meg.
- getEventsTags() => Csak azokat a tegeket jelenítjük meg, amelyek már hozzá vannak kapcsolva eseményekhez.
- searchOnPress()=> A szabadszavas és a teg alapú keresést valósítja meg aszerint, hogy a felhasználó melyiket szeretné alkalmazni, és akár a kettőt össze is tudja kombinálni.

- `removeDuplicateEvents(events:any[]) =>` Ez a metódus megakadályozza, hogy egy esemény véletlenül kétszere szerepeljen. Ennek folyamata az, hogy a bemeneti paraméterben kapott eseményeken végig megyünk, csinálunk egy key változót az esemény nevének és azonosítójának kombinációjával. Ellenőrizzük, hogy ezzel a kulccsal van már e esemény, és ha nincs, akkor ezt az eseményt hozzáadjuk a `uniqueEvents` tömbhöz, és ezt az egyedi eseményeket tartalmazó tömböt adjuk vissza.
- `resetSearch()` => Akkor fut le, amikor a felhasználó a szabadszavas keresésnél, az általa begépett szöveget kitörli, ennek eredményeképp visszatér az összes esemény. Erre azért van szükség, mert ez egyfajta dinamikusságot ad a keresésnek.
- `subscribeToEvent(event:any) =>` A felhasználó ezen keresztül tud feliratkozni egy eseményre. Bemenő paraméter maga az esemény.
- `unsubscribeFromEvent(data:any)=>` A felhasználó ezzel le tud iratkozni az eseményről. Bemenő paraméter maga az esemény.
- `getUserEvents()`=> A felhasználó által feliratkozott események, amelyeket a `userEvents` változóba mentjük el. Erre azért van szükség, mert az esemény kártyákon eszerint nézzük, hogy az adott felhasználó fel van e iratkozva, vagy sem.
- `isEventSubscribed(eventId:number):boolean=>` Ez a metódus egy igaz vagy hamis értékkel tér vissza, hogy a bemeneti paraméter azaz event id alapján a user fel van e iratkozva az eseményre.
- `hasUserExperience(eventId:number):boolean=>` Ellenőrizzük, hogy a belépett felhasználó írt e valamilyen kommentet az adott eseményre. Visszatérési érték egy boolean, ezáltal az oldalon megjelenik a komment vagy nem.
- `onTagSelect(tag:any, event:Event) =>` A felhasználó checkbox alapú formánál rákattint az egyik értékre (ez a tag) az event pedig magát a checkboxot figyeli, és ha be van pipálva, akkor az adott teg vagy tegek hozzáadódnak a `selectedTags` tömbhöz.
- `clearTagSelections()`=>Ez egy kényelmi funkció, hogy a felhasználóknak ne kelljen egyesével a tegeknél a pipákat eltüntetni.
- `closeOffcanvas()`=> Ha a felhasználó kisebb képernyőn van, és ha a szűrőpanelen keres, akkor keresés esetén (szabadszavas vagy teg alapú keresés) zárja be a szűrőpanel ablakot automatikusan.

card-Pagination:

Feladata, az események megjelenítésénél, hogy ne egyszerre az összes, hanem részekre bontva jelenjenek meg, azt az illúziót keltve, hogy különböző oldalon vannak.

Változók:

- @Input() totalItems:any => Szülő komponenstől fogja kapni az értéket, amely az oldalon megjelenő esemény darabszáma lesz.
- @Input() currentPage:any => Szülő komponenstől fogja kapni az értéket, amely minden esetben 1 lesz az elején.
- @Input() itemsPerPage:any => Szülő komponenstől fogja kapni az értéket, mi határozzuk meg, hogy egy oldalon hány darab elemet (eseményt) szeretnénk látni.
- @Output() onclick: EventEmitter<number> => Ez fogja küldeni az oldalszámot a szülő komponens felé, ezzel megvalósítva a lapozást.
- pages=>Ez fogja tárolni az oldalak számait (1,2,3...).

Metódusok:

- ngOnInit():void=> Amikor létrejön az oldal, lefut az calculatePages() metódus.
- pageClicked(page: number)=> Magát a lapozást valósítja meg.
- calculatePages()=> Kiszámolja, hogy mennyi oldalra törje az adatokat.
- ngOnChanges(changes: SimpleChanges):void=> Ha változik a totalItems vagy az itemsPerPage, akkor újból kiszámolja, hogy az adatokat az oldalon hány darab oldalba törje.
- getCurrentRoute():string=> Az aktuális útvonalat kapjuk vissza.

datepicker-range-popup

Feladata, mikor a felhasználó a naptár ikonra kattint, akkor jelenjen meg egy dinamikus dátumválasztó, amelyen ki tud választani dátumot, illetve dátum intervallumot.

Változók:

- calendar => Ez a változó tartalmazza az NgbCalendar által nyújtott szolgáltatásokat (pl: mai dátum megszerzése)
- formatter => Ez a változó tartalmazza a NgbDateParserFormatter-t, ami segít a dátumokkal való műveletek végrehajtásához.
- hoveredDate ::NgbDate | null => Ez a változó segít abban, hogy mikor a felhasználó egérrel szeretne dátum kiválasztani, akkor ráhúzva az egeret vizuálisan is látja, hogy mit jelöl ki illetve, ha már kijelölt egy kezdő dátumot és ki szeretné jelölni a vége dátumot, akkor a naptárban vizuálisan kejlölődnek azok a napok is, amelyek a

kezdő és a vég nap között vannak, ezzel segít a felhasználónak a dátumkiválasztásban.

- `fromDate :: NgbDate | null =>` Ez a változó tartalmazza a felhasználó által kijelölt kezdődátumot.
- `toDate :: NgbDate | null =>` Ez a változó tartalmazza a felhasználó által kijelölt végedátumot.

Metódusok:

- `onDateSelection (date:NgbDate) =>` Ez a metódus akkor fut le, mikor a felhasználó rákattint egy dátumra. A metódus figyeli, hogy ki van e választva kezdő érték (`fromDate`), hogy a vége dátum (`toDate`) későbbi dátum e mint a kezdő dátum (`fromDate`), illetve hogy vége dátum (`toDate`) egyáltalán ki van e választva.
- `isHovered(date:NgbDate) =>` Feladata, hogy vizuálisan kijelölve mutassa azokat a napokat, amelyek a kijelölt kezdődátum (`fromDate`) és a egérrel rámutatott, de még rá nem kattintott (`toDate`) között vannak.
- `isInside(date:NgbDate) =>` Feladata, hogy vizuálisan kijelölve mutassa azokat a napokat, amelyek a kijelölt kezdődátum (`fromDate`) és a egérrel rámutatott, és már rákattintott (`toDate`) között vannak.
- `isRange(date:NgbDate) =>` Ez a metódus valósítja meg a teljes dátumkiválasztás logikáját és vizualitását, felhasználva az `isInside()` és az `isHovered()` metódust.
- `validateInput(currentValue: NgbDate | null, input:string):NgbDate|null =>` Ez a metódus valósítja meg azt, hogy ellenőrzi a felhasználó által megadott input helyesen van e megadva, és ha nem, akkor a `currentValue` értéke érvényesül, ezzel megakadályozva hogy hiba lépjen fel.
- `setVisible():boolean =>` Mikor a felhasználó rákattint a naptár ikonra, akkor megjelenjen a dátumválasztó, vagy pedig tűnjön el.
- `clearSelection() =>` A felhasználó, ha kiválasztott kezdő és vége dátumot, akkor azoknak az értékét egy kattintással állítsa null-ra, ez a funkció a felhasználó kényelmét szolgálja.

dated-event

Feladata, hogy a felhasználó által kiválasztott eseményt egy új lapon megjelenjen részletesen, ahol látja az esemény bővebb információit, és kommentet tud hozzáfűzni az eseménnyel kapcsolatban.

Változók:

- `events:any` => Az összes eseményt tartalmazza, amelyet a backendtől kapunk.
- `tags:any` => Az összes teget tartalmazza, amit a backendtől kapunk.
- `detailedEventId:number|null` => Annak az eseménynek az id-ját tartalmazza, amire a felhasználó rákattintott.
- `eventDetails:any` => Az adott eseményhez tartozó összes adatot ebben a változóban tároljuk.
- `eventStartDateNull` => Mikor az backend felől érkezett esemény nem tartalmaz kezdő dátumot, akkor az ebben tárolt szöveg jelenik meg a programkártyán.
- `eventEndDateNull` => Mikor az backend felől érkezett esemény nem tartalmaz befejező dátumot, akkor az ebben tárolt szöveg jelenik meg a programkártyán.
- `eventStartTimeNull` => Mikor az backend felől érkezett esemény nem tartalmaz kezdő időpontot, akkor az ebben tárolt szöveg jelenik meg a programkártyán (ha egy üres string, akkor semmi).
- `eventEndDateNull` => Mikor az backend felől érkezett esemény nem tartalmaz befejező időpontot, akkor az ebben tárolt szöveg jelenik meg a programkártyán (ha egy üres string, akkor semmi).
- `userEvents:any` => Ebben a változóban tároljuk, a bejelentkezett felhasználó által feliratkozott eseményeket, és erre azért van szükség, mert ez alapján tudjuk azt megjeleníteni, hogy az adott felhasználó fel van e iratkozva, illetve ha fel van iratkozva, akkor tudjon kommentet fűzni az adott eseményhez.
- `user:any` => Ez a változó tárolja, a bejelentkezett felhasználó adatait, és erre azért van szükség, hogy tudjuk, ki az a user, aki bejelentkezett, és hogy egyáltalán be van e jelentkezve.
- `userCommentFromInput:any` => Ha a felhasználó valamilyen kommentet szeretne hozzáfűzni az által feliratkozott eseményhez, akkor itt tárolódik az.
- `userCommentFromApi:any` => Ha az adott felhasználónak már volt valamilyen kommentje az adott eseményhez, akkor azt megkapjuk a backendtől.
- `detailedEventTags:[]` => Azokat a tegeket tartalmazza, amelyek az adott eseményhez hozzá vannak kapcsolva.
- `activeSection:string` => ez a változó egy szöveget tartalmaz, ami azt befolyásolja, hogy a felhasználó mikor mit szeretne látni (esemény leírás, információk, vélemények), ha az esemény leírásra kattint, akkor az jelenik meg, ha utána az információkra kattint,

akkor az előző eltűnik és az információk jelennek meg. Ez dinamikusságot ad az oldalnak.

- `isNewExperienceVisible:boolean` => Ez a változó ahhoz kell, hogy megjelenítsük az oldalon a felhasználó kommentjét.
- `newExperienceDesabled:boolean` => Ez a változó ahhoz kell, hogy az input mezőt a felhasználó kitöltheti vagy sem.
- `editButtonVisible:boolean` => Ez a változó ahhoz kell, hogy a szerkesztő gomb látható legyen vagy sem (ha éppen ír egy eseményhez beszámolót akkor látható ez a gomb).
- `pencilIconDesabled:boolean` => Lehetővé teszi, hogy a szerkesztést jelölő ceruza ikon látható e vagy sem.
- `userAllExperience` => A felhasználó által megírt élménybeszámolóit tartalmazza.
- `sameEventTags` => A hasonló események tegeit tartalmazza.
- `relatedEventsByTags:any[]` => A címkék alapján a hasonló eseményeket tartalmazza.
- `hideNavbar` => A navbar eltüntetéséhez és megjelenítéséhez szükséges.

Metódusok:

- `getDataFromApi()` => Az összes esemény lekérése a backendtől, ami azért kell, hogy majd az összes esemény közül azt jelenítsük meg, amelyikre a felhasználó rákattintott.
- `ngAfterViewInit():void` => A navbar eltűnéséhez és megjelenítését biztosító logika.
- `detailedEvent(id:number | null)` => Annak az eseménynek a részleteit, amelyre a felhasználó rákattintott, megkapjuk a backendtől és megjelenítjük a felhasználónak.
- `formatUrl(webLink: string) :string` => A kártyákon, ahol valamilyen linket szeretnénk megjeleníteni, ott ha a link http-vel kezdődik akkor a http-t cserélje le <https://-re>
- `get sameEvents()` => Ez a getter olyan eseményeket dobna vissza, amelyek egyeznek a kiválasztott eseménnyel tegek alapján.
- `setActiveSection (section :string)` => beállítja az `activeSection` értékét.
- `getTags()` => Lekéri a backend-től az összes eseményt az oldal.
- `subscribeToEvent(event:any)` => A felhasználó ezen keresztül tud feliratkozni egy eseményre. Bemenő paraméter maga az esemény.
- `unsubscribeFromEvent(data:any)` => A felhasználó ezzel le tud iratkozni az eseményről. Bemenő paraméter maga az esemény.

- `getUserEvents()`=> A felhasználó által feliratkozott események, amelyeket a `userEvents` változóba mentjük el. Erre azért van szükség, mert az esemény kártyákon eszerint nézzük, hogy az adott felhasználó fel van e iratkozva, vagy sem.
- `isEventSubscribed(eventId:number):boolean`=> Ez a metódus egy igaz vagy hamis értékkel tér vissza, hogy a bemeneti paraméter azaz event id alapján a user fel van e iratkozva az eseményre.
- `getUserExperience()` => Ha volt már korábban kommentje az adott eseményhez a felhasználónak, akkor betöltődik a komment.
- `hasUserExperience(eventId:number):boolean`=> Ellenőrizzük, hogy a belépett felhasználó írt e valamilyen kommentet az adott eseményre. Visszatérési érték egy boolean, ezáltal az oldalon megjelenik a komment vagy nem.
- `showNewExperience()`=> Ha a felhasználó szeretné megjeleníteni a saját kommentjét.
- `cancelNewExperience()` => Ha meg szeretné szakítani az új élmény hozzáadása folyamatát a felhasználó, mikor rákattint a bezárás gombra.
- `updateExperience()`=> Ha a felhasználó módosítani szeretné a korábban megírt kommentjét.
- `editExperience()`=> Ez a metódus felelős azért, hogy a szerkesztés gomb láthatóvá váljon a felhasználó számára és akkor lesz látható, ha már volt korábban kommentje az adott eseménnyel kapcsolatban.
- `cancelUpdateExperience()` => Ha a felhasználó, mégsem szeretné módosítani a korábban írt hozzászólását az adott eseményhez, akkor a bezárás gombbal megszakíthatja ezt a folyamatot.
- `canActivateFeature(startDateStr:string)`=> Az élménybeszámoló gomb aktívvá válásának logikáját valósítja meg úgy, hogy megnézi hogy a jelenlegi dátum nagyobb-e, mint a kezdő dátuma az adott eseménynek.
- `getEventsTags()` => Csak azokat a tegeket jelenítjük meg, amelyek már hozzá vannak kapcsolva eseményekhez.
- `sameEventTagsWithoutDuplication()`=> Megakadályozza, hogy a hasonló események többször jelenjenek meg.
- `mapRelatedEventDetails()`=> Ez a metódus megszerzi, az adott részletes eseménnyel hasonló események adatait, a hasonló eseményeknél is lássuk a részleteket.
- `getShortUrl(url:string):string`=> A weblinkek rövidített megjelenítéséhez szükséges.

events-admin-list

Feladata, hogy az admin jogosultsággal rendelkező felhasználók ezen az oldalon fel tudjanak venni eseményeket és ha kell tudjanak rajtuk módosítani vagy törölni egy meghatározott eseményt.

Változók:

- newEventSuccess=> Új esemény felvétele ha hibával történik egy adat felvétele, akkor megjelenik a hibaüzenet.
- eventModifySuccess=> Már meglévő esemény valamely adatának helytelen módosítása esetén, a hibaüzenet megjelenik.
- erName:any=> Ha az esemény nevének megadását rontják el, akkor itt tárolódik a backend felől érkező hibaüzenet.
- erStartDate:any=> Ha az eseménynek a kezdő dátumának a megadását rontja el az admin joggal rendelkező felhasználó, akkor itt tárolódik a backend felől érkező hibaüzenet.
- erEndDate:any=> Ha az eseménynek a vége dátumának a megadását rontja el az admin joggal rendelkező felhasználó, akkor itt tárolódik a backend felől érkező hibaüzenet.
- erStartTime:any=> Ha az eseménynek a kezdő időpontjának a megadását rontja el az admin joggal rendelkező felhasználó, akkor itt tárolódik a backend felől érkező hibaüzenet.
- erEndTime:any => Ha az eseménynek a vége időpontjának a megadását rontja el az admin joggal rendelkező felhasználó, akkor itt tárolódik a backend felől érkező hibaüzenet.
- erModName:any=> Ha az esemény nevének hibás módosítása esetén itt tárolódik a backend felől érkező hibaüzenet.
- erModStartDate:any=> Ha az eseménynek a kezdő dátumának a módosítását rontja el az admin joggal rendelkező felhasználó, akkor itt tárolódik a backend felől érkező hibaüzenet.
- erModEndDate:any=> Ha az eseménynek a vége dátumának a módosítását rontja el az admin joggal rendelkező felhasználó, akkor itt tárolódik a backend felől érkező hibaüzenet.
- erModStartTime:any=> Ha az eseménynek a kezdő időpontjának a módosítását rontja el az admin joggal rendelkező felhasználó, akkor itt tárolódik a backend felől érkező hibaüzenet.
- erModEndTime:any => Ha az eseménynek a vége időpontjának a módosítását rontja el az admin joggal rendelkező felhasználó, akkor itt tárolódik a backend felől érkező hibaüzenet.

- `newEvent=>` Ez egy kulcs-érték párosból álló objektum típusú változó, amelyet egy form-ba illesztve az admin jogosultsággal rendelkező felhasználó kitölt, majd a mentés gombra kattintva ezt elküldjük a backend-nek, amely a kapott adatokat elmenti az adatbázisba.
- `events:any =>` Ebben tároljuk a már létező eseményeket, amelyeket megjelenítünk az oldalon.
- `editModeId: number | null =>` A meglévő események szerkesztésénél használjuk, mikor az admin jogosultságú felhasználó rákattint a szerkesztés gombra (melyet egy ceruza ikon imitál), akkor ez a változó megkapja a meghatározott eseménynek az id-ját. Ahol az `editModeId` megegyezik az adott esemény id-jával, ott megfog jelenni a szerkesztő tábla, amely biztosítani fogja a felhasználónak az átlátható adatmódosítást.
- `errModfyMsg: any =>` Meglévő események helytelen módosításakor fellépő hibaüzeneteket fogja tartalmazni, amit a backend felől fogunk kapni.
- `errNewEventMsg:any =>` Új esemény felvételénél, ha valamit helytelenül töltött ki a felhasználó, akkor a backend felől érkező hibaüzeneteket ebben a változóban tároljuk.

Metódusok:

- `getDataFormApi()` => A már meglévő eseményeket megkapjuk a backend-től és megjelenítjük a weblapon.
- `editRow(event:any)` => Itt történik meg az esemény id-nak átadása az `editModeId` változónak. Bemeneti paraméter maga az esemény, és abból fogjuk kivenni az id-t.
- `cancelEdit()` => Ha az admin joggal felhasználó mégsem szeretné módosítani az általa kiválasztott eseményt, azaz a módosítási folyamatot meg szeretné szakítani.
- `updateData(data:any)` => Az admin joggal felhasználó az általa kiválasztott eseménynek valamely adatát szeretné módosítani. Bemeneti paraméter maga az esemény, és abból fogjuk kivenni az id-t.
- `deleteData(data:any)` => Az admin joggal rendelkező felhasználó valamely már meglévő esemény szeretné eltávolítani az oldal adatbázisából. Bemeneti paraméter maga az esemény, és abból fogjuk kivenni az id-t.
- `newData()` => Új esemény felvételénél használjuk.

events-subscribed:

Feladata, a bejelentkezett felhasználó által az általa feliratkozott események megjelenítése.

Változók:

- currentPage=> Az oldalon belül a az események megjelenítésénél használjuk.
- itemsPerPage=> Az oldalon belül a az események megjelenítésénél használjuk. Egy oldalon hány darab eseménykártya lehet.
- eventStartDateNull=> Mikor az backend felől érkezett esemény nem tartalmaz kezdő dátumot, akkor az ebben tárolt szöveg jelenik meg a programkártyán.
- eventEndDateNull=> Mikor az backend felől érkezett esemény nem tartalmaz befejező dátumot, akkor az ebben tárolt szöveg jelenik meg a programkártyán.
- user:any => Ez a változó tárolja, a bejelentkezett felhasználó adatait, és erre azért van szükség, hogy tudjuk, ki az a user, aki bejelentkezett, és hogy egyáltalán be van e jelentkezve.
- selectedOption=> Itt tárolódik, hogy a felhasználó ABC sorrendben, vagy dátum szerinti sorrendben szeretné megjeleníteni a programokat.
- tags => Ebben tároljuk a backendtől megszerezett tegeket.
- selectedTags=> A felhasználó által kiválasztott tegeket ebben tároljuk, és ezt használva valósítjuk meg a teg alapú szűrést.
- tagSearch=> Ez egy bool, ami azt mondja meg, hogy éppen keresek e teg alapján vagy sem.
- tagsOfEvents:any[] => Csak azok a tegek vannak ebben a tömbben, amelyhez hozzá vannak kapcsolva események.
- attachedDatas:any[] =>getEventsTags() metódusnál használjuk szerepe, hogy tárolja a teg-esemény kapcsolatot.
- searchControl=> Ebben tárolódik a felhasználó által begépelt szavak, és amely a szabadszavas keresés megvalósításához szükséges.
- isSearch=> Ez egy bool, ami azt mondja meg, hogy éppen használom e a szabadszavas keresést vagy nem.
- commonSearchResults: Ebbe töltjük bele a searchTagResults vagy searchResults eseményeit, de ha egyszerre használom a kettő fajta keresést, akkor a kettőnek uniója töltődik bele ebbe a változóba.
- userEventsWithTags:any[]=> Ez tárolja, az eseményeket, és a hozzájuk tartozó összes címkét.

- `userEvents` => Ebben tárolódnak azok az események, amelyekre a bejelentkezett felhasználó feliratkozott.
- `userExperience:any` => Ez tartalmazza a felhasználó által írt kommenteket.

Metódusok:

- `getEventsTags()` => Csak azokat a tegeket jelenítjük meg, amelyek már hozzá vannak kapcsolva eseményekhez.
- `getUserExperience()` => Ha volt már korábban kommentje az adott eseményhez a felhasználónak, akkor betöltődik a komment.
- `changePage(page:number)` => Megvalósítja a lapozást az oldalon.
- `toSort()` => A weboldalon az eseményeknek az ABC- vagy dátum szerinti növekvő vagy csökkenő sorrendbe állítását valósítja meg.
- `searchOnPress()` => A szabadszavas és a teg alapú keresést valósítja meg aszerint, hogy a felhasználó melyiket szeretné alkalmazni, és akár a kettőt össze is tudja kombinálni.
- `resetSearch()` => Akkor fut le, amikor a felhasználó a szabadszavas keresésnél, az általa begépett szöveget kitörli, ennek eredményeképp visszatér az összes esemény. Erre azért van szükség, mert ez egyfajta dinamikusságot ad a keresésnek.
- `subscribeToEvent(event:any)` => A felhasználó ezen keresztül tud feliratkozni egy eseményre. Bemenő paraméter maga az esemény.
- `unsubscribeFromEvent(data:any)` => A felhasználó ezzel le tud iratkozni az eseményről. Bemenő paraméter maga az esemény.
- `getUserEvents()` => A felhasználó által feliratkozott események, amelyeket a `userEvents` változóba mentjük el. Erre azért van szükség, mert az esemény kártyákon eszerint nézzük, hogy az adott felhasználó fel van e iratkozva, vagy sem.
- `getUserEventsTags()` => Megkapjuk azokat a címkéket, amelyek hozzá vannak kapcsolva a feliratkozott eseményekhez.
- `isEventSubscribed(eventId:number):boolean` => Ez a metódus egy igaz vagy hamis értékkel tér vissza, hogy a bemeneti paraméter azaz event id alapján a user fel van e iratkozva az eseményre.
- `onTagSelect(tag:any, event:Event)` => A felhasználó checkbox alapú formnál rákattint az egyik értékre (ez a tag) az event pedig magát a checkboxot figyeli, és ha be van pipálva, akkor az adott teg vagy tegek hozzáadódnak a `selectedTags` tömbhöz.
- `clearTagSelections()` => Ez egy kényelmi funkció, hogy a felhasználóknak ne kelljen egyesével a tegeknél a pipákat eltüntetni.

- `get paginatedEvents():any[]=>` Ez a getter segít megvalósítani a lapozáshoz szükséges logikát, amelyet az összes eseményen alkalmazunk.
- `get paginatedSearchedEvents():any[]=>` Ez a getter segít megvalósítani a lapozáshoz szükséges logikát, amelyet a már leszűrt eseményeken (címkékkel vagy szabad szavasan keresett) alkalmazunk.
- `changePage(page:number)=>` Megvalósítja a lapozást az oldalon.
- `hasUserExperience(eventId:number):boolean=>` Ellenőrizzük, hogy a belépett felhasználó írt-e valamilyen kommentet az adott eseményre. Visszatérési érték egy boolean, ezáltal az oldalon megjelenik a komment vagy nem.
- `showSubscribedEventsAfterSearch(searchedEvents:any)=>` Szabad szavas illetve teg alapú keresés esetén azoknak az eseményeknek a megjelenítése, amelyekre fel van iratkozva a felhasználó.
- `navigateToEvent(eventId :number) =>` Ha be van jelentkezve a látogató, akkor a paraméterként kapott esemény azonosítója alapján átkerül a felhasználó az adott esemény részletes oldalára, ha nincs, akkor a bejelentkezés lapra kerül.
- `removeDuplicateEvents(events:any[]) =>` Ez a metódus megakadályozza, hogy egy esemény véletlenül kétszere szerepeljen. Ennek folyamata az, hogy a bemeneti paraméterben kapott eseményeken végig megyünk, csinálunk egy key változót az esemény nevének és azonosítójának kombinációjával. Ellenőrizzük, hogy ezzel a kulccsal van már-e esemény, és ha nincs, akkor ezt az eseményt hozzáadjuk a `uniqueEvents` tömbhöz, és ezt az egyedi eseményeket tartalmazó tömböt adjuk vissza.

events-tags-link

Feladat egy olyan felület biztosítása, ami admin jogosultsággal rendelkező felhasználó számára érthető el, és rajta össze tudja kapcsolni az eseményeket a hozzájuk legjobban illeszkedő tegekkel.

Változók:

- `searchIdControl=>` Az admin jogosultsággal rendelkező felhasználó, itt fog az esemény azonosítója alapján keresni.
- `searchNameControl=>` Az admin jogosultsággal rendelkező felhasználó, itt fog az esemény neve alapján keresni.
- `searchEventResults =>` Itt fogjuk tárolni a kereséskor megkapott eseményeket.

- `isEventSearch` => Ezt a változót arra használjuk, hogy éppen keresünk e eseményt vagy sem.
- `noEventSearchResults` => Feladata, hogy hamis érték esetén, ha nincs találat az eseményre, akkor megjelenik egy szöveg, ami felsorolja a lehetséges okokat. Igaz érték esetén pedig egy táblázat jelenik meg, amelyben az admin joggal rendelkező felhasználó hozzáadhat tegeket az adott eseményhez.
- `searchTerm:string` => Amikor tegekkel rendelkező események között szeretne a felhasználó keresni szabadszavasan, akkor az általa begépelt szöveget tartalmazza.
- `searchWithoutTagList:string` => Amikor tegekkel NEM rendelkező események között szeretne a felhasználó keresni szabadszavasan, akkor az általa begépelt szöveget tartalmazza.
- `tags:any` => A tegek tárolására használjuk.
- `selectedTags:number []` => Ebben a tömbben a kiválasztott tegnek az id-ját fogjuk használni, amit a html fájlban egy form-check-input form-ból nyerünk ki.
- `events:any` => Ebben tároljuk az összes eseményt.
- `attachedDatas:any` => A már összekapcsolt teg-esemény párost tartalmazza.
- `newAttachTagToEvent:any` => Ezt a változót arra használjuk, hogy tegeket kapcsoljunk eseményekhez a `newAttach()` metódusban.
- `showWithTag` => Ez igaz/hamis értékeket tárol, és arra használjuk, ha igaz, akkor az összepárosított teg-eseményeket jeleníti meg, ha hamis akkor összes eseményt jeleníti meg, amelyekhez még nincs kapcsolva címke.
- `eventsWithoutTags:any` => Ez az összes eseményt tartalmazza tegek nélkül, és arra használjuk, mikor a felhasználó direkt olyan eseményeket akarunk megjeleníteni, amikhez még nincs hozzáadva címke.

Metódusok:

- `searchWithTagTable()` => Ha a `searchTerm` üres azaz a felhasználó nem írt be semmit a keresőbe, akkor az összes teg-esemény megjelenik, ha pedig begépelt valamit, akkor azt leszűri a gép és ha van találat, akkor megjelenik az általa keresett teg-esemény/teg-események.
- `searchWithoutTagTable()` => Ha a `searchWithoutTagList` üres azaz a felhasználó nem gépelt be semmit, akkor az összes esemény megjelenik, amihez nem kapcsolódik teg. Ha gépelt valamit, akkor

az általa keresett esemény fog megjelenni, amelyhez nincs teg kapcsolva.

- `searchEvent()` => Ez a kereső biztosítja az admin joggal rendelkező felhasználó számára, hogy rákeressen egy eseményre id vagy név alapján, majd ha szeretne kapcsoljon hozzá egy vagy több teget.
- `getTags()` => Betölti az összes teget a tags változóba, melyeket akkor jelenítünk meg, mikor a felhasználó egy vagy több teget szeretne kötni az általa kiválasztott eseményhez.
- `getDataFromApi()` => Az összes esemény lekérése a backendtől.
- `updateSelectedTags(event:Event,tagId:number)` => Ez tárolja, hogy az admin joggal rendelkező felhasználó által kiválasztott címkéket/tegeket. Az event az a checkbox, mikor rákattintunk, tagId pedig a checkboxnál lévő tegnek az id-ja.
- `newAttach(eventId:number, selectedTags:number[])` => Ez a metódus valósítja meg a kiválasztott tegek egy adott eseményhez való kapcsolását. eventId az adott esemény id-ja, a selectedTags pedig azok a tegek, amelyeket a felhasználó kiválasztott.
- `getEventsTags()` => Ez a metódus feltölti az attachedDatas-t a már összekapcsolt esemény-teg elemekkel.
- `deleteTagFromEvent(data:any)` => A felhasználó által kiválasztott eseményről el tudja távolítani a tegeket. Bemeneti paraméter maga az esemény.
- `getEventsWithoutTag()` => `eventsWithoutTags`-et feltölti azokkal az eseményekkel, amelyekhez nincsenek kapcsolva tegek.

footer

Feladata, hogy az oldal alján a lábléceket, ahol a látogató láthatja az elérhetőségeinket a logónkat.

home

Ez a főoldal, itt a látogató ízelítőt kaphat arról, hogy milyen eseményekre tudna jelentkezni, ha regisztrál az oldalunkra.

Változók:

- `events:any` => Ez tartalmazza az összes eseményt, amelyeket a backend-től kérünk le.
- `randomEvents:any` => Ez olyan eseményeket tárol, ami véletlenszerűen kiválasztva az összes eseményből.

- user:any => Ebben tároljuk a bejelentkezett user adatait (id, adminjog, token). Ahoz használjuk, hogy tudjuk, hogy ha valaki interakcióba akar lépni valamivel, akkor az a személy egyáltalán be van e jelentkezve.
- eventStartDateNull => Ha a betöltött eseménynek nincs eventStartDate adata, akkor Állandó szöveg legyen a null érték helyett.
- eventEndDateNull => Ha a betöltött eseménynek nincs eventEndDate adata, akkor semmi ne jelenjen ott meg.
- userEvents:any => A bejelentkezett felhasználók által feliratkozott események megjelenítéséhez szükséges
- userExperience:any=> Ez tartalmazza a felhasználó által írt kommenteket.

Metódusok:

- getDataFromApi()=> Az események betöltése a weboldalra.
- navigateToEvent(eventId :number) => Ha be van jelentkezve a látogató, akkor a paraméterként kapott esemény azonosítója alapján átkerül a felhasználó az adott esemény részletes oldalára, ha nincs, akkor a bejelentkezés lapra kerül.
- getRandomEvents() => Ez sorsolja véletlenszerűen az eseményeket, melyeket megjelenítünk az oldalon.
- subscribeToEvent(data:any)=>A bejelentkezett felhasználó ezzel a metódussal tud feliratkozni az általa tetszetősnek tartott eseményre. A bemenő paraméter maga az esemény, amelyre fel szeretne iratkozni a felhasználó.
- unsubscribeFromEvent(data:any)=> A bejelentkezett felhasználó ezzel a metódussal tud leiratkozni arról az eseményről, amelyre már feliratkozott, de meggondolta magát.
- getUserEvents()=> A felhasználó által feliratkozott események, amelyeket a userEvents változóba mentjük el. Erre azért van szükség, mert az esemény kártyákon eszerint nézzük, hogy az adott felhasználó fel van e iratkozva, vagy sem.
- isEventSubscribed(eventId:number):boolean=> Ez a metódus egy igaz vagy hamis értékkel tér vissza, hogy a bemeneti paraméter azaz event id alapján a user fel van e iratkozva az eseményre.
- getUserExperience() => Ha volt már korábban kommentje az adott eseményhez a felhasználónak, akkor betöltődik a komment.
- hasUserExperience(eventId:number):boolean=> Ellenőrizzük, hogy a belépett felhasználó írt e valamilyen kommentet az adott

eseményre. Visszatérési érték egy boolean, ezáltal az oldalon megjelenik a komment vagy nem.

login

Feladata, hogy a látogató be tudjon jelentkezni az oldalunkra.

Változók:

- name => Ez tárolja a felhasználó által begépelte nevet.
- password=> Ez tárolja a felhasználó által begépelte jelszót.
- email=> Ez tárolja a felhasználó által begépelte emailt.
- mailRegError => Hibás bejelentkezés esetén true-ra változik, és megjeleníti a hibaüzenetet, amit a backend küld és a mailRegText-ben tárolódik.
- mailRegText => Ez tárolja a backend felől érkező hibaüzenetet, hibás bejelentkezés esetén.
- eyeIconToShowPassword=> true vagy false érték, amely a jelszó megjelenítéséhez szükséges
- toSwitchPasswordInputTypeDynamic=> text, vagy password szöveg, hogy az input típusát átalakítsuk szövegezzé, vagy jelszó típusá
- errorNameMessage => Hibás felhasználónév megadásánál a névre írja ki a hibaüzenetet.
- errorPasswordMessage => Hibás jelszó megadásánál a jelszóra írja ki a hibaüzenetet.
- unknownMessageBool => Ha hibás a jelszó és/vagy a felhasználónév, akkor ez az érték true.
- loginError => Ha a felhasználó nem tölti ki a felhasználó nevet és a jelszót, vagy valamelyiket elrontja, akkor true az értéke.

Metódusok:

- visiblePassword() => Ettől válik elolvashatóvá a jelszómező.
- loginUserOnLaravel() => A felhasználó bejelentkeztetése az oldalra
- logoutUserWithLaravel() => A felhasználó kijelentkeztetése, ez azért kell, hogy a laravel által generált tokenet kitöröljük az adatbázisból.
- setVisible():boolean => Az oldal látogatója úgy kattint rá egy eseményre, hogy nincs bejelentkezve, akkor ez az érték true lesz, és az oldal tetején az a szöveg lesz látható, hogy jelentkezzen be vagy regisztráljon az oldalunkra.

modal

Feladata, hogy felugró ablakok jelenjenek meg üzenetekkel, mikor a felhasználó interakcióba lép pl: a feliratkozás gombbal. Ez egy minta, amelyet minden komponensnél fel lehet használni.

Változók:

- @Input() title: string => Ez a változó a szülőosztálytól kapja az értékét, amely string típusú.
- @Input() content: string => Ez a változó a szülőosztálytól kapja az értékét, amely string típusú.
- @Input() closeButtonText: string => Ez a változó a szülőosztálytól kapja az értékét, amely string típusú.
- @Input() actionButtonText: string => Ez a változó a szülőosztálytól kapja az értékét, amely string típusú.
- @Input() actionButtonClass: string => Ez a változó a szülőosztálytól kapja az értékét, amely string típusú.
- @Output() action => Ez egy kimeneti eseményt definiál a komponensből, amelyre a szülő komponens fel tud iratkozni.

Metódusok:

- onActionClick() => Ez a függvény akkor fut le, amikor a modalban az **akciógombra** (pl. "OK", "Törlés") kattint a felhasználó.

navbar

Feladata a weboldalon a navigációs sáv biztosítása.

Változók:

- user:any => A bejelentkezett user adatainak mentése (név, jogosultság, token).

Metódusok:

- logoutUserWithLaravel() => A navigációs sávon ha a felhasználó a kijelentkezés gombra kattint, akkor kijelentkezik az oldalról (adatai törlődnek a localStorage-ból) és az adatbázisból törlésre kerül a token-je.

- collapseNavbar()=> Ha túl kicsi a képernyő/kijelző, akkor a navigációs sávot összezsugorolja.

registration

Feladata, hogy a látogató be tudjon regisztrálni az oldalunkra.

Változók:

- name => Ez tárolja a felhasználó által begépelte nevet.
- password=> Ez tárolja a felhasználó által begépelte jelszót.
- confirmPassword => Ez tárolja a felhasználó által begépelte jelszót. Ez ahhoz kell, hogy megvizsgáljuk, hogy a felhasználó jól írta be a jelszavát, és ha van elütés valamelyikben, akkor a regisztráció nem történjen meg.
- email=> Ez tárolja a felhasználó által begépelte emailt.
- mailRegError=> boolean típusú változó, mely akkor használjuk, ha a regisztrációnál a felhasználó valamit rosszul tölt ki.
- mailRegText=> A backend válaszait tartalmazza.
- eyeIconToShowPassword=> true vagy false érték, amely a jelszó megjelenítéséhez szükséges.
- toSwitchPasswordInputTypeDynamic=> text, vagy password szöveg, hogy az input típusát átalakítjuk szövegesre, vagy jelszó típusúvá.
- registrationSuccess => A regisztráció sikeressége esetén ad vissza true vagy false értéket.
- erEmail:any=> Hibásan megadott email cím esetén tartalmazza a backend hibaüzenetét.
- erName:any=> Hibásan megadott felhasználó név esetén tartalmazza a backend hibaüzenetét.
- erPassword:any=> Hibásan megadott jelszó esetén tartalmazza a backend hibaüzenetét.

Metódusok:

- isValidSignUp() => Megakadályozza, hogy a felhasználó úgy tudjon regisztrálni az oldalra, hogy nincs kitöltve az email, név, jelszó, illetve ha különbözik a jelszó és a jelszó megerősítés értéke
- userRegistration() => A felhasználó regisztrálása a weboldalunkra. Erről email-ben értesítést kap a felhasználó.
- visiblePassword() => Ettől válik elolvashatóvá a jelszó mező.

tags-list

Feladata, hogy az admin jogosultsággal rendelkező felhasználók láthassák, hogy milyen tegek vannak, és képesek legyenek új tegeket felvenni, vagy meglévőt módosítani vagy törölni.

Változók:

- tags: any => Ez a változó tartalmazza az összes teget, amelyet a backend-től kapunk válaszul.
- newTagSuccess=> Boolean típusú változó, mely a hibaüzenetek megjelenítéséhez használjuk (newData() metódusban).
- erName:=> Ha hibásan adjuk meg a címke nevét, akkor a backend hiba üzenetét tartalmazza.
- erGroup:any=> Ha hibásan adjuk meg a címke típusát, akkor a backend hiba üzenetét tartalmazza
- tagModSuccess=> Boolean típusú változó, amelyet az updateData() metódusban használunk fel, és sikeres módosítás esetén bezárja a szerkesztő ablakot.
- erModName:=> Ha hibásan módosítjuk a címke nevét, akkor a backend hiba üzenetét tartalmazza.
- erModGroup:any=> Ha hibásan módosítjuk a címke típusát, akkor a backend hiba üzenetét tartalmazza
- newTagItem => A form használatánál szükséges ez az objektum típusú változó, ami két fajta értéket tud felvenni, egy name mező, ami a teg neve lesz, és egy group-ot, ami pedig a teg típusa lesz.
- editModeId: number| null => A meglévő tegek szerkesztésénél használjuk, mikor az admin jogosultságú felhasználó rákattint a szerkesztés gombra (melyet egy ceruza ikon imitál), akkor ez a változó megkapja a meghatározott tegnek az id-ját. Ahol az editModeId megegyezik az adott teg id-jával, ott megfog jelenni a szerkesztő tábla, amely biztosítani fogja a felhasználónak az átlátható adatmódosítást.

Metódusok:

- getTags() => Tegek betöltése a weboldalra, hogy azokkal az admin jogosultsággal rendelkező felhasználó műveleteket tudjon végezni.
- updateData(data:any)=> A meglévő tegeken módosításokat végre tudjon hajtani a felhasználó. Bemeneti paraméter maga a teg, melynek adatait módosítani szeretné.

- `deleteData(data:any)` => A felhasználó által kiválasztott teg törlődik az adatbázisból. Bemeneti paraméter maga a teg, amelyet törölni szeretne.
- `newData()` => A felhasználó a form kitöltésével egy új teget regisztrál az adatbázisban.
- `editRow(tag:any)` => Itt történik meg a teg id-nak átadása az `editModeId` változónak. Bemeneti paraméter maga a teg, és abból fogjuk kivenni az id-t.
- `cancelEdit()` => Ha az admin joggal felhasználó mégsem szeretné módosítani az általa kiválasztott teget, azaz a módosítási folyamatot meg szeretné szakítani.

toast-messages

Feladata, hogy a felhasználókat felugró ablakokkal közölje az általuk végzett műveletek sikerességéről (siker üzenetek, hibaüzenetek)

user-settings

Feladata, hogy a felhasználókat felugró ablakokkal közölje az általuk végzett műveletek sikerességéről (siker üzenetek, hibaüzenetek)

users

Feladata, hogy az admin joggal rendelkező felhasználók láthassák, hogy kik vannak regisztrálva az oldalra, azokon műveleteket tudjanak végezni (ha superadmin lépett be az oldalra, akkor az ő esetében admin jogosultságot adjon az adott felhasználónak).

Változók:

- `users:any` => Az oldalra regisztrált felhasználók betöltése a weboldalra.
- `editModeId: number | null` => A meglévő felhasználók szerkesztésénél használjuk, mikor az superadmin/admin jogosultságú felhasználó rákattint a szerkesztés gombra (melyet egy ceruza ikon imitál), akkor ez a változó megkapja a meghatározott tegnek az id-ját. Ahol az `editModeId` megegyezik az adott teg id-jával, ott megfog jelenni a szerkesztő tábla, amely biztosítani fogja a felhasználónak az átlátható adatmódosítást.

- selectDisabled => Ez a változó ahhoz kell, hogy egyes inputokat módosíthatatlanná állítsuk. (pl: ha a felhasználó csak sima admin, akkor ő nem adhat admin szintű jogosultságot másoknak.)

Metódusok:

- getAllUsers() => Az összes regisztrált felhasználó adatainak betöltése.
- updateData(data: any) => A kiválasztott felhasználón módosítást lehet végrehajtani (felhasználónév módosítás).
- isUserSuperadmin()=> Ellenőrizzük, hogy a bejelentkezett felhasználó superadmin- e és ha igen akkor admin jogosultságot adhat felhasználóknak.
- deleteData(adat:any) => Felhasználó törlése a weboldalról.
- editRow(user:any) => Itt történik meg az felhasználók id-nak átadása az editModeId változónak. Bemeneti paraméter maga az felhasználó, és abból fogjuk kivenni az id-t.
- cancelEdit() => Ha az admin joggal/superadmin joggal rendelkező felhasználó mégsem szeretné módosítani az általa kiválasztott felhasználónak a jogosultságát, azaz a módosítási folyamatot meg szeretné szakítani.

Fejlesztési lehetőségek

- A felhasználó ha elfelejtette a jelszavát, akkor kérjen új jelszót, amivel beléphet, és azt megváltoztathatja.
- A felhasználónak legyen egy beállítások lap, ahol tudja módosítani a jelszavát.
- A felhasználó kapjon értesítést email-ben a közelgő eseményekről, amelyekre feliratkozott.
- Lehessen a date-picker komponenst használni az időpontokra történő szűrés során.
- User-settings komponens teljes értékű megvalósítása.
- Megvalósítani a datepicker-range-popup komponens használatát.
- Emlékeztető beállítása a kiválasztott programokhoz.
- bevezetni annak lehetőségét, hogy a felhasználók is tudjanak eseményt rögzíteni. Privát eseményt maguk számára, publikus eseményt mindenki számára megjelenítve. A publikus eseményeket az adminok moderálnák, hogy ne kerülhessen ki a közös területre bármilyen tartalom.
- Felhasználók kapcsolódásának lehetősége: pl. privát események megosztása

- programok értékelésének lehetősége számértékkel + szövegesen. Ez minden felhasználó számára látható lenne
- események helyszínének megtekinthetősége térképen megjelenítve
- Személyes beállítások oldalon legyen lehetőség rögzíteni a felhasználó érdeklődési köreit (címkéket választhatna ki), amely alapján időről időre értesítést kapna a legfrissebb eseményekről.
- bevonni az események szervezőit, hogy a saját eseményeiket ők rögzíthessék a felületre
- az oldalon keresztül történő tényleges jelentkezés az adott eseményre (pl sport események, versenyek)

Backend

Általános működés:

A REST API http kéréseket fogad, melyek tartalmazzák a műveletekhez szükséges megfelelő adatokat. A kényes műveletek végpontjai védettek, autentikáció szükséges a használatukhoz.

Ilyenek a kijelentkezés, az új diák felvétele, diák adatainak módosítása, diák törlése, új csoport felvétele, csoport módosítása, csoport törlése. A regisztráció, a bejelentkezés, a diákok és csoportok lekérdezésének végpontjai publikusak, ezek használatához autentikáció nem szükséges.

Az adatokat Json formátumban fogadja és feldolgozza. A vezérlést kontrollerek valósítják meg, minden adatkezelési csoportnak (autentikáció, diák, csoport) külön kontrollere van, itt történik az adatfeldolgozás.

A kontrollerek modellekkel vannak kapcsolatban, amelyek az adatkezelésért felelősek. Minden adatkezelési csoportnak (felhasználó, diák, csoport) külön modellje van, itt történik az adatok adatbázisból kiolvasása, illetve a az adatok kiírása adatbázisba.

A modellek adatbázis táblákkal vannak kapcsolatban, melyek az adatok tárolásáért felelősek.

Az adatbázis táblák adatait a modellek kezelik.

Env. fájlhoz szükséges adatok

Adatbázis

beállítások:

DB_CONNECTION=mysql

DB_HOST=127.0.0.1

DB_PORT=3306

DB_DATABASE=ProgramvalasztoDB

DB_USERNAME=root

DB_PASSWORD=

E-mail küldéshez

a

beállítások:

MAIL_MAILER=smtp

```
MAIL_SCHEME=smtp
MAIL_HOST=smtp.gmail.com
MAIL_PORT_DEFAULT=465
MAIL_PORT_ALT=587
MAIL_USERNAME=programvalaszto@gmail.com
MAIL_PASSWORD=pniuzbnningducdw
MAIL_FROM_ADDRESS="programvalaszto@gmail.com"
MAIL_FROM_NAME="{APP_NAME}"
```

Útvonalak

Ezek a fájlok a Laravel alkalmazás API és webes útvonalaiért felelnek meg. Az alábbi kódbestand a api.php fájl tartalmát képviseli, amely a RESTful API végpontokat definiálja.

Útvonalak Részletes Elemzése

Regisztráció és Bejelentkezés:

```
Route::post("/register", [ UserController::class, "register" ]);
```

Cél: Új felhasználó regisztrálása.

```
Route::post("/login", [UserController::class, "login"]);
```

Cél: Felhasználó bejelentkezése.

Email ellenőrzés:

```
Route::get('email/verify/{id}/{hash}', [VerificationController::class,
'verify'])->name('verification.verify');
```

Cél: Email címek ellenőrzése a regisztrációs folyamat során.

Hitelesített Felhasználói útvonalak:

Ezek az útvonalak csak hitelesített felhasználók (akik beléptek) számára elérhetők a auth:sanctum middleware-en keresztül.

Kilépés:

```
Route::post("/logout", [ UserController::class, "logOut" ]);
```

Keresések:

Események, tagok és felhasználók keresése:

searchEvent, searchTags, searchUser funkciók.

Felhasználók kezelése:

Felhasználók listázása, módosítása, törlése:

getUsers, modifyUser, destroyUser funkciók.

Események kezelése:

Események létrehozása, módosítása, törlése:

newEvent, modifyEvent, destroyEvent.

Címkék kezelése:

Címkék létrehozása, módosítása, törlése:

newTag, modifyTag, destroyTag.

Csoportok és Események Tag-ekhez kapcsolása:

Eseményekhez és címkékhez kapcsolódó funkciók (attachTags, detachTag, getEventsByTag, getTagsByEvents).

Feliratkozások kezelése:

Feliratkozások, mint például eseményekhez való feliratkozás, módosítás és leiratkozás.

Felhasználó és Címkék kapcsolat:

subUserTag és unsubUserTag funkciók, amelyek lehetővé teszik, hogy a felhasználók címkéket kapjanak vagy azokról leiratkozzanak

Migration

Az adatbázis-séma definíciói az database/migrations könyvtárban található migrációs fájlokban találhatók.

Műveletek:

up() Metódus:

A metódus célja a táblák létrehozása.

down() Metódus:

A metódus célja a migráció visszaállítása.

create_cache_table

Fájl Neve: 0001_01_01_000001_create_cache_table.php

Feladat:

A migrációs fájl célja, hogy létrehozza a cache és cache_locks adatbázistáblákat, amelyek a Laravel alkalmazás cache (gyorsítótár) rendszerének működéséhez szükségesek. Ezek a táblák tárolják a gyorsítótárba mentett adatokat, valamint az adatlopás és a konkurens írások elkerülése érdekében a zárat.

Definiált Adatbázis Táblák:

cache Tábla:

key (string, primary key): A gyorsítótárban tárolt adat kulcsa. Ez biztosítja, hogy minden tárolt adat egyedi legyen.

value (mediumText): A tárolt adat értéke, amely bármilyen típusú lehet, ezért mediumText típust kapott, lehetővé téve a nagyobb adatok tárolását.

expiration (integer): Ez a mező tárolja, hogy meddig érvényes a gyorsítótárazott adat, azaz mikor jár le a cache.

cache_locks Tábla:

key (string, primary key): A zár kulcsa, ami biztosítja, hogy minden zárolt erőforrás egyedi.

owner (string): A zár sajátos tulajdonosa, amely segíthet az azonosításban és a zárolás kezelésében, különösen, ha több felhasználó vagy folyamat próbál hozzáférni az azonos adat erőforráshoz.

expiration (integer): A zár időtartama, amely jelzi, hogy meddig érvényes a lock. Ezt az értéket súlyosbítja a cache-elésnél alkalmazott időkorlát, segítve elkerülni a halmozódást vagy a holt időszakokat.

create_personal_access_tokens_table

Fájl

Neve:

2025_01_09_122629_create_personal_access_tokens_table.php

Feladat:

Ez a migrációs fájl a personal_access_tokens nevű adatbázistábla létrehozásáért felelős, amely a felhasználók számára személyes hozzáférési tokenek kezelésére szolgál. Ez a funkció alapvető eleme az

API hitelesítésének, lehetővé téve a felhasználók számára, hogy biztonságosan hozzáférjenek az alkalmazás erőforrásaihoz.

Definiált Adatbázis Tábla:

personal_access_tokens Tábla:

id (bigint, primary key): Automatikusan generált egyedi azonosító minden egyes token számára.

tokenable_id (bigint): A felhasználó azonosítója (vagy bármilyen más tokenhez kapcsolódó entitás), amelyhez a token tartozik. Ez a mező a polymorphic kapcsolat egyik része.

tokenable_type (string): A tokenhez tartozó entitás típusa (például User vagy Admin). Ez a mező is része a polymorphic kapcsolatnak.

name (string): A személyes token neve, amely lehetővé teszi a felhasználóknak, hogy azonosítsák a token funkcióját vagy célját.

token (string, 64 karakter hosszú, unique): A token értéke. Ez az érték egyedi, és használható az API hitelesítéséhez.

abilities (text, nullable): A tokenhez kapcsolódó képességeket leíró mező, amely opcionális. Ez lehetővé teszi a finomhangolt jogosultságok megadását a tokenhez.

last_used_at (timestamp, nullable): Az utolsó időpont, amikor a tokent használták. Ez segít a token érvényességi idejének és aktivitásának nyomon követésében.

expires_at (timestamp, nullable): A token lejáratási idejét tárolja. Ez lehetővé teszi a tokenek automatikus érvénytelensé válását egy előre meghatározott időpont után.

created_at (timestamp): A token létrehozásának időpontja.

updated_at (timestamp): A token utolsó frissítésének időpontja.

create_users_table

Fájl Neve: 2025_01_09_151734_create_users_table.php

Feladat:

Ez a migrációs fájl a `users`, `password_reset_tokens`, és `sessions` adatbázistáblák létrehozásáért felelős, amelyek a Laravel alkalmazás felhasználói kezelésének, jelszó-visszaállításának, és munkamenet-kezelésének alapját képezik.

Definiált Adatbázis Táblák:

`users` Tábla:

`id` (`bigint`, `primary key`): Automatikusan generált egyedi azonosító minden felhasználó számára.

`name` (`string`): A felhasználó neve, amely lehetővé teszi a felhasználó azonosítását.

`email` (`string`): A felhasználó e-mail címe, amely szintén egyedi.

`email_verified_at` (`timestamp`, `nullable`): Az e-mail cím érvényesítési dátumának tárolására szolgáló mező.

`password` (`string`): A felhasználó jelszava, amelyet hash-elte formában kell tárolni a biztonság érdekében.

`rememberToken` (`string`, `nullable`): A "memorizálás" funkcióhoz szükséges token, amely lehetővé teszi a felhasználó számára, hogy bejelentkezve maradjon.

`created_at` (`timestamp`): A felhasználó rekord létrehozásának időpontját tárolja.

`updated_at` (`timestamp`): A felhasználói rekord utolsó frissítésének időpontját tárolja.

`password_reset_tokens` Tábla:

`email` (`string`, `primary key`): A felhasználó e-mail címe, amelyhez a jelszó-visszaállítási token tartozik. Ez a mező egyedi, és biztosítja az azonosítást.

`token` (`string`): A jelszó-visszaállításához szükséges token, amelyet a felhasználónak meg kell adnia.

`created_at` (`timestamp`, `nullable`): A token létrehozásának időpontját tároló mező, amely `nullable`, tehát nem kötelező.

sessions Tábla:

id (string, primary key): Minden munkamenet egyedi azonosítója, amely egyedi azonosítót biztosít a session kezeléséhez.

user_id (foreignId, nullable, index): A felhasználó azonosítója, akihez a munkamenet kapcsolódik. Nullable, azaz nem minden session szükségszerűen kapcsolódik felhasználóhoz.

ip_address (string, 45, nullable): A felhasználó IP-címének tárolására szolgál, azaz nem kötelező.

user_agent (text, nullable): A felhasználó eszközének vagy böngészőjének azonosítása érdekében gyűjtött adat, amely nullable.

payload (longText): A munkamenethez kapcsolódó adatokat tartalmazó mező.

last_activity (integer, index): Az utolsó aktivitás időpontja, amely lehetővé teszi a munkamenetek időkorlátjának kezelését.

create_events_table

Fájl Neve: 2025_01_09_151748_create_events_table.php

Feladat:

Ez a migrációs fájl az events nevű adatbázistábla létrehozásáért felelős, amely események adatait tárolja. A táblázat funkciója, hogy egyszerűsíti az események kezelését és lekérdezését az alkalmazásban, lehetővé téve különböző események információinak tárolását.

Definiált Adatbázis Tábla:

events Tábla:

id (bigint, primary key): Automatikusan generált egyedi azonosító minden esemény számára.

name (string): Az esemény neve, amely azonosítja az eseményt.

startDate (date, nullable): Az esemény kezdésének dátuma.

endDate (date, nullable): Az esemény befejezésének dátuma.

startTime (time, nullable): Az esemény kezdési időpontja.

endTime (time, nullable): Az esemény befejezési időpontja.

description (text, nullable): Az esemény leírása, amely tartalmazza a részletes információkat.

image (string, nullable): Az eseményhez kapcsolódó kép URL-je.

locationName (string, nullable): Az esemény helyszínének neve.

locationcountry (string, nullable): Az esemény helyszínének ország neve.

address (string, nullable): Az esemény pontos címe, ami opcionális.

state (integer, nullable): Az esemény állapotát jelölő mező, amely megadhatja az esemény státuszát.

weblink (string, nullable): Az eseményhez tartozó weboldal hivatkozása, amely opcionális.

latitude (decimal, nullable): Az esemény földrajzi szélességi koordinátája, amely a helyszín pontos lokalizálásához szükséges.

longitude (decimal, nullable): Az esemény földrajzi hosszúsági koordinátája, amely szintén a helyszín azonosításához tartozik.

gpx (string, nullable): A GPX fájl hivatkozása, amennyiben az eseményhez kapcsolódik.

subscribed (unsignedInteger, default: 0): Az eseményre feliratkozott felhasználók számát tároló mező, amely alapértelmezetten 0-ra van állítva.

created_at (timestamp): Az esemény rekord létrehozásának időpontja.

updated_at (timestamp): Az esemény rekord utolsó frissítésének időpontja.

create_tags_table

Fájl Neve: 2025_01_09_151810_create_tags_table.php

Feladat:

Ez a migrációs fájl a tags nevű adatbázistábla létrehozásáért felelős, amely a címkék (tags) kezelését szolgálja az alkalmazásban. A címkék csoportosítást és kategorizálást tesznek lehetővé az események, tartalmak vagy egyéb objektumok számára.

Definiált Adatbázis Tábla:

tags Tábla:

id (bigint, primary key): Automatikusan generált egyedi azonosító minden címkehez.

name (string): A címke neve, amely alapján az azonosítást végzik.

group (string): A címke csoportja, amely lehetővé teszi a címkek hierarchikus szerkezetbe való rendezését, vagy csoportosítását.

created_at (timestamp): A címke rekord létrehozásának időpontját tárolja.

updated_at (timestamp): A címke rekord utolsó frissítésének időpontját tárolja.

create_subscriptions_table

Fájl Neve: 2025_01_09_152058_create_subscriptions_table.php

Feladat:

Ez a migrációs fájl a subscriptions nevű adatbázistábla létrehozásáért felelős, amely a felhasználók eseményekre való feliratkozásait tárolja. A táblázat lehetővé teszi a kapcsolat kialakítását a felhasználók és az események között, valamint kiegészítő információk gyűjtését a feliratkozásokkal kapcsolatban.

Definiált Adatbázis Tábla:

subscriptions Tábla:

id (bigint, primary key): Automatikusan generált egyedi azonosító minden feliratkozáshoz.

users_id (foreignId): A felhasználó azonosítója, aki feliratkozott az eseményre. Külső kulcs a users táblára, amely a felhasználók adatait tartalmazza. A constrained('users') megköveteli, hogy ez a mező a users táblában lévő id mező értékével egyezzen meg. Az onDelete('cascade') biztosítja, hogy ha egy felhasználót törölnek, akkor a kapcsolódó feliratkozások is automatikusan törölődnek.

events_id (foreignId): Az esemény azonosítója, amelyre a felhasználó feliratkozott. Külső kulcs az events táblára. A constrained('events') garantálja, hogy az értéke a events táblában lévő id mező értékével

egyezzen meg. Az onDelete('cascade') biztosítja, hogy ha egy eseményt törölnek, akkor a kapcsolódó feliratkozások is automatikusan törölődnek.

comment (text, nullable): Opcionális mező, amely lehetővé teszi a felhasználó számára, hogy további megjegyzést fűzzön a feliratkozásához.

sub_date (date): A feliratkozás/ részvétel dátuma; ez a mező a feliratkozás időpontját rögzíti.

created_at (timestamp): A rekord létrehozásának időpontját tárolja.

updated_at (timestamp): A rekord utolsó frissítésének időpontját tárolja.

create_favorites_table

Fájl Neve: 2025_01_09_152141_create_favorites_table.php

(Nincs alkalmazva)

Feladat:

Ez a migrációs fájl a favorites nevű adatbázistábla létrehozásáért felelős, amely a felhasználók kedvenc eseményeit tárolja. A táblázat célja, hogy lehetővé tegye a felhasználók számára, hogy megjelöljék azokat az eseményeket, amelyeket különösen fontosnak tartanak, és így könnyen hozzáférhessenek ezekhez később.

Definiált Adatbázis Tábla:

favorites Tábla:

users_id (foreignId): A felhasználó azonosítója, aki a kedvenc eseményeket hozzáadja. Külső kulcs a users táblára, amely a felhasználók adatait tárolja.

events_id (foreignId): Az esemény azonosítója, amely a felhasználó kedvence. Külső kulcs az events táblára, amely az esemény részleteit tartalmazza.

created_at (timestamp): A rekord létrehozásának időpontját tárolja.

updated_at (timestamp): A rekord utolsó frissítésének időpontját tárolja.

create_comments_table

Fájl Neve: 2025_01_09_152155_create_comments_table.php

(Nincs alkalmazva)

Feladat:

Ez a migrációs fájl a comments nevű adatbázistábla létrehozásáért felelős, amely a felhasználók által az eseményekre írt megjegyzéseket és értékeléseket tárolja. A táblázat célja, hogy lehetővé tegye a felhasználók számára, hogy visszajelzést adjanak az eseményekről, ezzel javítva az interakciót és a közösségi élményt.

Definiált Adatbázis Tábla:

comments Tábla:

users_id (foreignId): A felhasználó azonosítója, aki a megjegyzést írta. Külső kulcs, amely a users táblát hivatkozza.

events_id (foreignId): Az esemény azonosítója, amelyre a megjegyzés vonatkozik. Külső kulcs, amely az events táblát hivatkozza.

comment (text, nullable): A felhasználó által írt megjegyzés szövege.

rating (integer): Az eseményre adott értékelés.

created_at (timestamp): A rekord létrehozásának időpontját tárolja.

updated_at (timestamp): A rekord utolsó frissítésének időpontját tárolja.

create_eventtags_table

Fájl Neve: 2025_01_09_152210_create_eventtags_table.php

Feladat:

Ez a migrációs fájl az eventtags nevű adatbázistábla létrehozásáért felelős, amely az események és címkék közötti kapcsolatokat tárolja. Az eventtags tábla célja, hogy lehetővé tegye az események címkézését, így segítve a könnyebb keresést és kategorizálást az események között.

Definiált Adatbázis Tábla:

eventtags Tábla:

events_id (foreignId): Az esemény azonosítója, amely a címkéhez kapcsolódik. Külső kulcs az events táblára, amely az események adatait tárolja. A constrained() módszerrel létrehozott külső kulcs automatikusan hivatkozik az events tábla megfelelő mezőjére.

tags_id (foreignId): A címke azonosítója, amely az eseményhez kapcsolódik. Külső kulcs a tags táblára, amely a címkék adatait tárolja. A constrained() metódus szintén biztosítja, hogy a hivatkozott adat érvényes legyen.

created_at (timestamp): A rekord létrehozásának időpontját tárolja.

updated_at (timestamp): A rekord utolsó frissítésének időpontját tárolja.

admin_to_users_table

Fájl Neve: 2025_02_19_162330_add_admin_to_users_table.php

Feladat:

Ez a migrációs fájl a users adatbázistáblához egy új mező, az admin, hozzáadásáért felelős. Ez a mező célja, hogy jelezze, egy adott felhasználó adminisztrátor-e vagy sem.

Definiált Adatbázis Tábla:

users Tábla:

admin (integer): Ez a mező egy egész számot tárol, amely a felhasználói jogosultságok szintjét jelzi. Az 1-es és 2-es érték jelenti azt, hogy a felhasználó adminisztrátor vagy szuper adminisztrátor, míg egy 0 érték a normál felhasználót jelenthet.

create_tag_subscriptions_table

Fájl Neve: 2025_02_23_105012_create_tag_subscriptions_table.php

Feladat:

Ez a migrációs fájl a tag_subscriptions nevű adatbázistábla létrehozásáért felelős, amely a felhasználók címke-feliratkozásait tárolja. Célja, hogy a felhasználók kedvenc címkéikhez való kapcsolódását nyomon kövesse, lehetővé téve számukra, hogy értesítést kapjanak az adott címkéhez kapcsolódó eseményekről.

Definiált Adatbázis Tábla:

tag_subscriptions Tábla:

users_id (foreignId): A felhasználó azonosítója, aki feliratkozott a címkére. Külső kulcs a users táblára, amely a felhasználói adatokat tárolja.

`tags_id` (foreignId): Az a címke azonosítója, amelyre a felhasználó feliratkozott. Külső kulcs a tags táblára, amely a címkék adatait tárolja.

`created_at` (timestamp): A feliratkozás időpontjának tárolására szolgál.

`updated_at` (timestamp): A feliratkozás frissítésének időpontját tárolja.

add_deleted_at_to_users_table

Fájl Neve: 2025_04_06_074012_add_deleted_at_to_users_table.php

Feladat:

Ez a migrációs fájl a users adatbázistáblához egy `deleted_at` mező hozzáadására szolgál, amely lehetővé teszi a lágy törléseket (soft deletes) az alkalmazásban.

Definiált Adatbázis Tábla:

users Tábla:

`deleted_at` (timestamp): Ez a mező tárolja a törlés időpontját, ha a felhasználót lágyan törölték. Ha az érték NULL, akkor a felhasználó aktív, míg ha van értéke, akkor ez azt jelzi, hogy a felhasználót törölték, de az adatainak megőrzése érdekében a rendszerben maradt.

Seeder

DatabaseSeeder

Feladata: A DatabaseSeeder osztály a Laravel adatbázis-seedelője, amely lehetővé teszi, hogy egyszerűen hozzassunk létre és tölthessünk fel alapvető adatokat az adatbázisban. Az osztályban meghatározott adatok alapvető felhasználói rekordokat tartalmaznak, amelyek a fejlesztés során referenciaként vagy tesztelésre használhatók.

Tulajdonságok és Metódusok:

`public function run(): void`

Leírás: Ez a metódus futtatja az adatok betöltéséhez szükséges kódot. A `$table->insert([..])` hívások segítségével rekordokat adhatunk hozzá az adatbázishoz.

Implementáció:

Felhasználói rekordok beszúrása: Két rekordot helyez a users táblába:

Az admin felhasználót és a superadmin felhasználót hasonló attribútumokkal, de magasabb jogosultsági szinttel.

Model

User

Feladata: A felhasználói adatok kezelésére és autentikációjára szolgál. Ez az osztály tartalmazza a felhasználók tulajdonságait és kapcsolatait, lehetővé téve a Laravel rendszerében történő hitelesítést és adatkezelést.

Használat: Az User osztály a Laravel autentikációs rendszere révén lehetővé teszi a felhasználók létrehozását, frissítését, törlését, valamint a kapcsolódó műveleteket, mint például a jelszókezelést és a szerepkörök ellenőrzését.

Tulajdonságok és Metódusok:

Traits:

HasFactory: Lehetővé teszi a gyári metódusok használatát a felhasználói modell előállításához.

Notifiable: Lehetővé teszi, hogy a felhasználó értesítéseket kapjon (pl. e-mail, SMS).

HasApiTokens: Lehetővé teszi API tokenek létrehozását a felhasználók számára.

SoftDeletes: Lehetővé teszi a lágy törlés (soft delete) használatát, azaz a rekordok nem kerülnek azonnal törlésre, hanem csak „törölt” állapotba kerülnek.

protected \$fillable:

Leírás: Definiálja azokat az attribútumokat, amelyeket tömegesen hozzáadhatunk (mass assignable), például a felhasználó neve, e-mail címe, jelszava és admin jogosultsága.

Tartalmazza:

'name'

'email'

'password'

'admin'

protected \$hidden:

Leírás: Megadja azokat az attribútumokat, amelyeket nem szeretnénk megjeleníteni a JSON sorosítás során (pl. API válaszokban).

Tartalmazza:

'password'

'remember_token'

protected function casts(): array:

Leírás: A modellek attribútumainak típushashálását definiálja.

Tartalmazza:

'email_verified_at' => 'datetime': Ez az attribútum dátumként van definiálva.

'password' => 'hashed': A jelszó automatikusan hashelt formában lesz tárolva.

public function subscriptions():

Leírás: A felhasználóhoz tartozó feliratkozások (Subscription) lekérdezésére szolgál.

Kimeneti adatok: A hasMany relációt visszaadó lekérdezés, amely megadja, hogy a felhasználónak több feliratkozása lehet.

public function tags(): BelongsToMany:

Leírás: A felhasználó és a címkék (Tag) közötti kapcsolatok kezelésére szolgál.

Kimeneti adatok: A belongsToMany relációt visszaadó lekérdezés a tag_subscriptions pivot táblán keresztül, megadva a users_id és tags_id kulcsokat, valamint a withTimestamps() metódussal biztosítja a létrehozási és frissítési időbélyegzők mentését.

Event

Feladata: Az események reprezentálására és kezelésére szolgáló modell. Az események lehetnek rendezvények, aktivitások, vagy bármilyen más

típusú esemény, amelyhez felhasználók, megjegyzések, címkék és feliratkozások kapcsolódhatnak.

Használat: Az Event osztály segítségével az alkalmazások képesek kezelni az eseményeket, valamint a kapcsolódó műveleteket, mint például a résztvevők nyilvántartását, a hozzászólások gyűjtését és a rendezvények kategorizálását.

Tulajdonságok és Metódusok:

Traits:

HasFactory: Lehetővé teszi a gyári metódusok használatát az események egyszerű előállításához.

Kapcsolatok:

public function users()

Leírás: Visszaadja a felhasználókat, akik kapcsolódnak az eseményhez.

Kimeneti adatok: belongsToMany relációt visszaadó lekérdezés, amely lehetővé teszi, hogy több felhasználó is részt vegyen egy eseményen.

public function comments()

Leírás: Visszaadja az eseményhez tartozó megjegyzéseket.

Kimeneti adatok: hasMany relációt visszaadó lekérdezés, amely megadja, hogy az eseményhez több hozzászólás is tartozhat.

public function tags()

Leírás: Visszaadja a címkéket, amelyek az eseményhez kapcsolódnak.

Kimeneti adatok: belongsToMany relációt visszaadó lekérdezés a eventtags pivot táblán keresztül.

Pivot tábla: eventtags

Felhasználó ID: events_id

Címke ID: tags_id

public function subscriptions()

Leírás: Visszaadja az eseményhez kapcsolódó feliratkozásokat.

Kimeneti adatok: hasMany relációt visszaadó lekérdezés, amely megadja, hogy az eseményhez több feliratkozás is tartozhat.

Subscription

Feladata: A felhasználók eseményekre való feliratkozásainak reprezentálására és kezelésére szolgáló modell. Az eseményekhez való feliratkozás lehetővé teszi a felhasználók számára, hogy nyomon követhessék az érdeklődési köreiket és részvételüket a különböző eseményeken.

Használat: Az Subscription osztály fontos szerepet játszik az események és felhasználók közötti kapcsolat kezelésében, lehetővé téve, hogy egy felhasználó több eseményre is feliratkozhatson, és információt tarthasson az feliratkozásával kapcsolatban, például észrevételeket és a feliratkozás dátumát.

Tulajdonságok és Metódusok:

Traits:

HasFactory: Lehetővé teszi a gyári metódusok használatát az feliratkozások egyszerű előállításához.

protected \$fillable:

Leírás: Megadja azokat az attribútumokat, amelyeket tömegesen hozzáadhatunk (mass assignable).

Tartalmazza:

'users_id': Az feliratkozó felhasználó azonosítója.

'events_id': Az esemény azonosítója, amelyre a felhasználó feliratkozott.

'comment': Az feliratkozáshoz kapcsolódó megjegyzés.

'sub_date': Az feliratkozás dátuma.

Kapcsolatok:

public function user()

Leírás: Visszaadja a felhasználót, aki az feliratkozást létrehozta.

Kimeneti adatok: belongsTo relációt visszaadó lekérdezés, amely lehetővé teszi, hogy egy feliratkozáshoz egy felhasználó tartozzon.

public function event()

Leírás: Visszaadja az eseményt, amelyhez az feliratkozás tartozik.

Kimeneti adatok: belongsTo relációt visszaadó lekérdezés, amely lehetővé teszi, hogy az feliratkozáshoz egy esemény tartozzon.

Egyedi kulcs: events_id - ez a mező jelöli meg az esemény azonosítóját.

Comment

(Nincs alkalmazva)

Feladata: A felhasználók által az eseményekhez fűzött megjegyzések reprezentálására és kezelésére szolgáló modell. A megjegyzések lehetővé teszik a felhasználók számára, hogy kifejezzék véleményüket, kérdéseiket, vagy bármilyen információt oszthassanak meg az adott eseménnyel kapcsolatban.

Használat: Az Comment osztály segítségével az alkalmazások képesek nyilvántartani az eseményekhez fűzött megjegyzéseket, amelyeket felhasználók adhatnak hozzá, ezzel növelve az interaktivitást és a felhasználói élményt.

Tulajdonságok és Metódusok:

Traits:

HasFactory: Lehetővé teszi a gyári metódusok használatát a megjegyzések egyszerű előállításához.

Kapcsolatok:

public function user()

Leírás: Visszaadja a felhasználót, aki a megjegyzést írta.

Kimeneti adatok: belongsTo relációt visszaadó lekérdezés, amely lehetővé teszi, hogy egy megjegyzéshez egy felhasználó tartozzon.

Használat: Ez a kapcsolat lehetővé teszi a felhasználói információk elérését a megjegyzésekből (pl. a megjegyzést író felhasználó neve vagy profilja).

public function event()

Leírás: Visszaadja az eseményt, amelyhez a megjegyzés tartozik.

Kimeneti adatok: belongsTo relációt visszaadó lekérdezés, amely lehetővé teszi, hogy egy megjegyzéshez egy esemény tartozzon.

Használat: Ez a kapcsolat lehetővé teszi a megjegyzés eseményhez való kapcsolódásának biztosítását, így könnyen lekérdezhető, hogy mely esemény alatt írták a megjegyzést.

Resource

User

Feladata: A felhasználói adatok (például az azonosítók, név, e-mail cím és adminisztrátori státusz) JSON formátumban történő átalakítása a Laravel API válaszain belül. Ez az osztály lehetővé teszi a felhasználói adatok strukturált megjelenítését és helyes kezelést.

Metódusok:

toArray(Request \$request): array

Leírás: Átalakítja a felhasználói erőforrást egy tömbbé (array), amely JSON válaszként visszaadható.

Bejövő paraméterek:

\$request: Az aktuális HTTP kérés, amely lehetővé teszi a további kontextus megadását.

Kimeneti adatok:

array: Az átalakított adatok tömbje. A következő mezőket tartalmazza:

"id": A felhasználó azonosítója (integer).

"name": A felhasználó neve (string).

"email": A felhasználó e-mail címe (string).

"admin": Boolean érték, amely jelzi, hogy a felhasználó adminisztrátor-e (boolean).

Event

Feladata: Az események adatait (például az azonosítót, nevet, időpontokat, leírást és egyéb részleteket) JSON formátumban történő átalakítása a Laravel API válaszaiban. Ez az osztály lehetővé teszi az események strukturált megjelenítését egy API-n keresztül.

Metódusok:

toArray(Request \$request): array

Leírás: Átalakítja az esemény erőforrást egy tömbbé (array), amely JSON válaszként visszaadható.

Bejövő paraméterek:

\$request: Az aktuális HTTP kérés, amely lehetővé teszi a további kontextus megadását.

Kimeneti adatok:

array: Az átalakított adatok tömbje, amely a következő mezőket tartalmazza:

"id": Az esemény azonosítója (integer).

"name": Az esemény neve (string).

"startDate": Az esemény kezdő dátuma (string).

"endDate": Az esemény befejezésének a dátuma (string).

"startTime": Az esemény kezdési időpontja (formázott string).

"endTime": Az esemény befejezésének az időpontja (formázott string).

"description": Az esemény leírása (string).

"image": Az eseményhez tartozó kép URL-je (string).

"locationName": Az esemény helyszínének neve (string).

"locationcountry": Az esemény helyszínének országa (string).

"address": Az esemény címe (string).

"state": Az esemény állapota (string).

"weblink": Az esemény weboldala (string).

"latitude": Az esemény földrajzi szélessége (float).

"longitude": Az esemény földrajzi hosszúsága (float).

"gpx": GPX fájl hivatkozás (string).

"subscribed": Boolean érték, amely jelzi, hogy mennyi felhasználó van feliratkozva az eseményre.

formatTime(?string \$time): ?string

Leírás: Segédmetódus az időpontok formázására. Ellenőrzi, hogy az időpont érvényes formátumban van-e, és ha igen, a H:i formátumban adja vissza.

Bejövő paraméterek:

\$time: Az időpont, amelyet formázni kell. Nullable (lehet null).

Kimeneti adatok:

string|null: A formázott időpont string formátumban, vagy null, ha az időpont üres vagy érvénytelen.

Tag

Feladata: A címke adatait (például az azonosítót, nevet és csoportot) JSON formátumban történő átalakítása a Laravel API válaszaiban. Ez az osztály lehetővé teszi a címkék egyszerű és strukturált megjelenítését egy API-n keresztül.

Metódusok:

toArray(Request \$request): array

Leírás: Átalakítja a címke erőforrást egy tömbbé (array), amely JSON válaszként visszaadható.

Bejövő paraméterek:

\$request: Az aktuális HTTP kérés, amely lehetővé teszi a további kontextus megadását.

Kimeneti adatok:

array: Az átalakított adatok tömbje, amely a következő mezőket tartalmazza:

"id": A címke azonosítója (integer).

"name": A címke neve (string).

"group": A címke csoportja (string).

Comment

(Nincs alkalmazva)

Feladata: A megjegyzés adatait (például a felhasználó azonosítóját, az esemény azonosítóját, a megjegyzést és a minősítést) JSON formátumban történő átalakítása a Laravel API válaszaiban. Ez az osztály lehetővé teszi a megjegyzések egyszerű és strukturált megjelenítését egy API-n keresztül.

Metódusok:

toArray(Request \$request): array

Leírás: Átalakítja a megjegyzés erőforrást egy tömbbé (array), amely JSON válaszként visszaadható.

Bejövő paraméterek:

\$request: Az aktuális HTTP kérés, amely lehetővé teszi a további kontextus megadást.

Kimeneti adatok:

array: Az átalakított adatok tömbje, amely a következő mezőket tartalmazza:

'user_id': A megjegyzést író felhasználó azonosítója (integer).

'event_id': Az esemény azonosítója, amelyhez a megjegyzés kapcsolódik (integer).

'comment': A megjegyzés szövege (string).

'rating': A megjegyzéshez tartozó minősítés, ha van (integer vagy null).

Kontrollerek

Controller

Feladata az admin jogosultság kezelése.

Metódusok:

authorizeAdmin() –a felhasználó jogkörének leellenőrzését végzi

kimenő adatok: nincs

bejövő paraméterek: \$user – felhasználó adatai

kimenő adatok: nincs

ResponseController

Feladata: A válaszok kezelése az API-nak, beleértve a sikeres és hibás válaszok küldését, valamint az adminisztratív jogosultságok ellenőrzését.

Metódusok:

sendResponse(\$data, \$message)

Leírás: Sikeres API válasz küldésére szolgál. A megadott adatot és üzenetet JSON formátumban adja vissza.

Bejövő paraméterek:

\$data: Az API válaszban visszaadandó adat. Ez lehet bármilyen típusú, amely a JSON formátumba konvertálható.

\$message: A válaszhoz tartozó üzenet, amely informálja a felhasználót a kérés eredményéről.

Kimenő adatok: Visszatér a JSON válasz, amely tartalmazza a következőket:

"success": true

"data": A megadott adat.

"message": A visszaküldött üzenet.

HTTP Státusz Kód: 200 OK

sendError(\$error, \$errorMessages = [], \$code = 404)

Leírás: Hibás API válasz küldésére szolgál. Megadja a hibaüzenetet és opcionálisan további hibaüzeneteket is.

Bejövő paraméterek:

\$error: A hiba üzenete, amely informálja a felhasználót a kérés problémáiról.

\$errorMessages: Opcionális. További hibák részletezése, amelyeket tömb formájában adhatunk meg.

\$code: Opcionális. A HTTP státusz kód, amely alapértelmezés szerint 404.

Kimenő adatok: Visszatér a JSON válasz, amely tartalmazza a következőket:

"success": false

"message": A megadott hibaüzenet.

("errorMessage": Ha vannak további hibaüzenetek, azok kerülnek ide.)

HTTP Státusz Kód: A megadott kód (alapértelmezetten 404).

adminAuth()

Leírás: Ellenőrzi a felhasználó admin jogosultságait. Ha a felhasználó adminisztrátor (admin jogkör 2), automatikusan engedélyezi a hozzáférést.

Bejövő paraméterek: Nincs.

Kimenő adatok: Ha a felhasználó nem rendelkezik megfelelő admin jogosultsággal, hibaüzenetet küld, ellenkező esetben nincs kimenet.

Hibaüzenetek: Ha a felhasználó nem felel meg a jogosultságoknak, visszatér a `sendError` metódust használva a következővel:

"Azonosítási hiba": Hibaüzenet.

"Nincs jogosultság": További információ a problémáról.

HTTP Státusz Kód: 401 Unauthorized.

VerificationController

Feladata: Az e-mail címek ellenőrzésének kezelése a felhasználók regisztrációjához. A controller lehetővé teszi a felhasználók számára az e-mail címük megerősítését egy kód vagy hash segítségével.

Metódusok:

`verify(Request $request, $id, $hash)`

Leírás: Ellenőrzi a felhasználó e-mail címét a megadott azonosító (`$id`) és hash (`$hash`) alapján.

Bejövő paraméterek:

`$request`: A HTTP kérés, amely tartalmazhat további adatokat.

`$id`: A felhasználó azonosítója, amely alapján az adatbázisból betöltjük a felhasználót.

`$hash`: A felhasználó e-mail címének titkosított formája, amelyet a belépési folyamat során használnak az e-mail ellenőrzéséhez.

Kimenő adatok:

Redirect: A felhasználót átirányítja a bejelentkezési oldalra, és különböző üzeneteket jelenít meg az eljárás eredményeként:

Ha a hash nem egyezik az e-mail cím SHA1 hash értékével: 'Érvénytelen ellenőrző link.'

Ha a felhasználó már igazolta az e-mail címét: 'Az e-mail már le van igazolva.'

Ha az e-mail cím sikeresen igazolva lett: 'Az e-mail sikeresen igazolva.'

HTTP Státusz Kód: 302 Found (mivel a felhasználót átirányítjuk).

UserController

Feladata felhasználók lekérése, regisztrációja, bejelentkeztetése, kijelentkeztetése, módosítása, törlése, keresése, fel és leiratkozása

Metódusok:

getUsers()

Leírás: Visszaküldi az aktív felhasználók listáját az adataikkal

Bejövő paraméterek: bearer token autentikációhoz

Kimenő adatok: Aktív felhasználók

register(RegisterRequest \$request)

Leírás: Regisztrál egy új felhasználót, amennyiben a megadott adatok megfelelnek a RegisterRequest által előírt feltételeknek.

Bejövő paraméterek:

"name": Felhasználónév.

"email": E-mail cím.

"password": Jelszó.

"confirm_password": Jelszó megerősítése.

Kimenő adatok: Nincs. (Sikeres regisztráció esetén e-mailt küld a felhasználónak.)

login(LoginRequest \$request)

Leírás: Ellenőrzi a felhasználó azonosítását név/email és jelszó alapján. Sikeres autentikáció esetén létre hoz egy token, amelyet tárol az adatbázisban, és visszaadja.

Bejövő paraméterek:

"name": Felhasználónév vagy e-mail.

"password": Jelszó.

Kimenő adatok: A sikeres bejelentkezés után visszaadott token és üzenet.
logout()

Leírás: Kijelentkezteti a felhasználót, és törli a kiválasztott tokent az adatbázisból.

Bejövő paraméterek: Bearer token az autentikációhoz.

Kimenő adatok: A sikeres kijelentkezés üzenete.

modifyUser(ModifyUserRequest \$request)

Leírás: Módosítja a felhasználó adatait, amennyiben a felhasználó jogosult az adatok módosítására.

Bejövő paraméterek:

"id": Felhasználó azonosítója.

"name": Felhasználónév.

"email": E-mail cím.

"password" (opcionális): Új jelszó.

"admin" (opcionális): Jogosultságok módosítása.

Kimenő adatok: Módosított felhasználó adatai vagy hibaüzenet.

destroyUser(int \$id)

Leírás: Törli a felhasználót az adatbázisból, amennyiben a felhasználó jogosult a törlésre.

Bejövő paraméterek:

int \$id: A törlendő felhasználó azonosítója.

Kimenő adatok: Törölt felhasználó adatai vagy hibaüzenet.

searchUser(LoginRequest \$request)

Leírás: Keres felhasználókat a megadott név alapján.

Bejövő paraméterek:

"query": Keresési kifejezés.

Kimenő adatok: A keresett felhasználók listája.

subUserTag(Request \$request)

Leírás: Hozzáad címkéket a felhasználóhoz.

Bejövő paraméterek:

"tags": Hozzáadni kívánt címkék azonosítói.

Kimenő adatok: A felhasználó új címkéivel együtt visszaadott üzenet.

unsubUserTag(Request \$request)

Leírás: Eltávolít címkéket a felhasználótól.

Bejövő paraméterek:

"tags": Eltávolítandó címkék azonosítói.

Kimenő adatok: Eltávolított címkék esetén visszaadott üzenet.

EventController

Feladata: Az események kezelésének központosítása az API-n belül, beleértve az események létrehozását, lekérdezését, módosítását, törlését, keresését és címkékkel való kapcsolat kezelését.

Metódusok:

getEvents()

Leírás: Lekéri az összes eseményt az adatbázisból.

Bejövő paraméterek: Nincs.

Kimenő adatok: JSON válasz, amely tartalmazza az összes eseményt EventResource formájában, és a következő üzenetet: "Események betöltve".

HTTP Státusz Kód: 200 OK

newEvent(EventRequest \$request)

Leírás: Új esemény létrehozása az API-n keresztül, ha a felhasználó admin jogosultsággal rendelkezik.

Bejövő paraméterek:

\$request: A HTTP kérés, amely tartalmazza az esemény adatait.

Kimenő adatok: JSON válasz, amely a létrehozott eseményt tartalmazza EventResource formájában, és a következő üzenetet: "Esemény létrehozva".

HTTP Státusz Kód: 201 Created

modifyEvent(EventRequest \$request)

Leírás: Meglévő esemény módosítása az API-n keresztül, ha a felhasználó admin jogosultsággal rendelkezik.

Bejövő paraméterek:

\$request: A HTTP kérés, amely tartalmazza az esemény módosítandó adatait.

Kimenő adatok: JSON válasz a módosított eseménnyel EventResource formájában, vagy hibaüzenet, ha az esemény nem létezik.

HTTP Státusz Kódok:

200 OK (ha a módosítás sikeres)

406 Not Acceptable (ha az esemény nem található)

destroyEvent(int \$id)

Leírás: Esemény törlése az API-n keresztül.

Bejövő paraméterek:

\$id: Az esemény azonosítója, amelyet törölni kell.

Kimenő adatok: JSON válasz a törölt eseményről, vagy hibaüzenet, ha az esemény nem létezik.

HTTP Státusz Kódok:

200 OK (ha a törlés sikeres)

406 Not Acceptable (ha az esemény nem található)

getEventId(EventRequest \$request)

Leírás: Lekéri egy esemény azonosítóját az esemény nevének megadásával.

Bejövő paraméterek:

\$request: A HTTP kérés, amely tartalmazza az esemény nevét.

Kimenő adatok: Az esemény azonosítója.

HTTP Státusz Kód: 200 OK

searchEvent(Request \$request)

Leírás: Események keresése a név, leírás vagy egyéb paraméterek alapján.

Bejövő paraméterek:

\$request: A HTTP kérés, amely tartalmaz egy keresési lekérdezést.

Kimenő adatok: JSON válasz az illeszkedő események gyűjteményével EventResource formájában.

HTTP Státusz Kód: 200 OK

attachTags(Request \$request, \$eventId)

Leírás: Címkék hozzáadása egy konkrét eseményhez.

Bejövő paraméterek:

\$request: A HTTP kérés, amely tartalmazza a címkék azonosítóit.

\$eventId: Az esemény azonosítója, amelyhez a címkék csatolandók.

Kimenő adatok: JSON válasz a frissített eseményről, vagy hibaüzenet, ha a címkék nem találhatók.

HTTP Státusz Kódok:

200 OK (ha a címkék sikeresen hozzáadva)

404 Not Found (ha az esemény vagy a címkék nem találhatók)

detachTag(\$eventId, \$tagId)

Leírás: Címke eltávolítása a megadott eseményről.

Bejövő paraméterek:

\$eventId: Az esemény azonosítója.

\$tagId: A címke azonosítója, amelyet el kell távolítani.

Kimenő adatok: JSON válasz a frissített eseményről, vagy hibaüzenet, ha az esemény nem található.

HTTP Státusz Kódok:

200 OK (ha a címke eltávolítás sikeres)

404 Not Found (ha az esemény nem található)

getTagsByEvents(int \$eventId)

Leírás: Lekéri az eseményhez tartozó címkéket.

Bejövő paraméterek:

\$eventId: Az esemény azonosítója.

Kimenő adatok: JSON válasz a címkékkel, vagy üzenet, ha nincs címke társítva.

HTTP Státusz Kódok:

200 OK (ha az eseményhez címkék társítva találhatók)

404 Not Found (ha az esemény nem található)

getEventsWithTags()

Leírás: Összes esemény lekérdezése címkékkel együtt.

Bejövő paraméterek: Nincs.

Kimenő adatok: JSON válasz az eseményekkel, vagy üzenet, ha nincsenek címkékkel ellátott események.

HTTP Státusz Kódok:

200 OK (ha vannak események címkékkel)

200 OK (ha nincsenek események, üzenettel)

TagController

Feladata: A címkék kezelésének központosítása az API-n belül, beleértve a címkék létrehozását, lekérdezését, módosítását, törlését, keresését és címkékhez kapcsolódó események lekérdezését.

Metódusok:

getTags()

Leírás: Lekéri az összes címkét az adatbázisból.

Bejövő paraméterek: Nincs.

Kimenő adatok: JSON válasz, amely tartalmazza az összes címkét TagResource formájában, és a következő üzenetet: "Címkék betöltve".

HTTP Státusz Kód: 200 OK

newTag(TagRequest \$request)

Leírás: Új címke létrehozása az API-n keresztül, ha a felhasználó admin jogosultsággal rendelkezik.

Bejövő paraméterek:

\$request: A HTTP kérés, amely tartalmazza a címke adatait (név, csoport).

Kimenő adatok: JSON válasz a létrehozott címkével TagResource formájában, és a következő üzenetet: "Címke kiírva".

HTTP Státusz Kód: 201 Created

modifyTag(TagRequest \$request)

Leírás: Meglévő címke módosítása az API-n keresztül, ha a felhasználó admin jogosultsággal rendelkezik.

Bejövő paraméterek:

\$request: A HTTP kérés, amely tartalmazza a módosítandó címke adatait (azonosító, név, csoport).

Kimenő adatok: JSON válasz a módosított címkéről TagResource formájában, vagy hibaüzenet, ha a címke nem létezik.

HTTP Státusz Kódok:

200 OK (ha a módosítás sikeres)

406 Not Acceptable (ha a címke nem található)

destroyTag(int \$id)

Leírás: Címke törlése az API-n keresztül.

Bejövő paraméterek:

\$id: A címke azonosítója, amelyet törölni kell.

Kimenő adatok: JSON válasz a törölt címkéről TagResource formájában, vagy hibaüzenet, ha a címke nem létezik.

HTTP Státusz Kódok:

200 OK (ha a törlés sikeres)

406 Not Acceptable (ha a címke nem található)

getTagId(TagRequest \$request)

Leírás: Lekéri egy címke azonosítóját a címke nevének megadásával.

Bejövő paraméterek:

\$request: A HTTP kérés, amely tartalmazza a címke nevét.

Kimenő adatok: A megtalált címke azonosítója.

HTTP Státusz Kód: 200 OK

searchTag(TagRequest \$request)

Leírás: Címkék keresése a név alapján.

Bejövő paraméterek:

`$request`: A HTTP kérés, amely tartalmaz egy keresési lekérdezést.

Kimenő adatok: JSON válasz az illeszkedő címkék gyűjteményével `TagResource` formájában.

HTTP Státusz Kód: 200 OK

`getGroups()`

Leírás: Visszaadja a címkecsoportokat, amiket a rendszer kezelésére használ.

Bejövő paraméterek: Nincs.

Kimenő adatok: JSON válasz a címkecsoportok listájával, és a következő üzenettel: "Csoportok betöltve".

HTTP Státusz Kód: 200 OK

`getEventsByTag(int $tagId)`

Leírás: Lekéri az eseményeket, amelyekhez a megadott címke kapcsolódik.

Bejövő paraméterek:

`$tagId`: A címke azonosítója, amelyhez az eseményeket lekérdezzük.

Kimenő adatok: JSON válasz az események gyűjteményével `EventResource` formájában, vagy üzenet, ha nincsenek események a címkéhez társítva.

HTTP Státusz Kódok:

200 OK (ha vannak események a címkéhez)

404 Not Found (ha a címke nem található)

SubscriptionController

Feladata: A feliratkozások kezelése az API-n belül, beleértve a felhasználók eseményekre történő feliratkozását, leiratkozását és a feliratkozások adatainak kezelését.

Metódusok:

`getSubscriptions()`

Leírás: Lekéri az aktuális felhasználó összes feliratkozását, az eseményekkel együtt.

Bejövő paraméterek: Nincs.

Kimenő adatok: JSON válasz a feliratkozások listájával, valamint egy üzenet: "Feliratkozások betöltve."

HTTP Státusz Kód: 200 OK

subscribe(SubscribeRequest \$request)

Leírás: Feliratkozás egy eseményre. Admin felhasználóknem iratkozhatnak fel.

Bejövő paraméterek:

\$request: A HTTP kérés, amely tartalmazza az events_id-t (az esemény azonosítója) és opcionálisan egy comment-et (kommentár).

Kimenő adatok: JSON válasz a létrehozott feliratkozással, valamint a következő üzenet: "Sikeres föliratkozás."

HTTP Státusz Kódok:

201 Created (ha a feliratkozás sikeres)

403 Forbidden (ha az admin felhasználó próbálkozik)

404 Not Found (ha az esemény nem található)

400 Bad Request (ha a felhasználó már fel van iratkozva)

updateSubscription(Request \$request, \$eventId)

Leírás: Eseményre vonatkozó feliratkozás frissítése, ahol a comment (élménybeszámoló) mező megváltoztatható.

Bejövő paraméterek:

\$request: A HTTP kérés, amely tartalmazza az opcionális comment-et.

\$eventId: Az esemény azonosítója, amire frissíteni szeretnénk a feliratkozást.

Kimenő adatok: JSON válasz a frissített feliratkozással, valamint a következő üzenet: "Sikeresen frissített feliratkozás."

HTTP Státusz Kódok:

200 OK (ha a frissítés sikeres)

404 Not Found (ha a felhasználó nincs feliratkozva vagy az esemény nem található)

403 Forbidden (ha az esemény kezdete nem járt le)

unsubscribe(\$eventId)

Leírás: Feliratkozás törlése az adott eseményről.

Bejövő paraméterek:

\$eventId: Az esemény azonosítója, amelyről le szeretnénk iratkozni.

Kimenő adatok: JSON válasz nullával (vagy üres válasz), és a következő üzenet: "Sikeres leiratkozás."

HTTP Státusz Kódok:

200 OK (ha a leiratkozás sikeres)

404 Not Found (ha a felhasználó nincs feliratkozva az eseményre)

Request

RegisterRequest

Feladata: A HTTP kérés érvényesítése a felhasználói regisztráció során. Biztosítja, hogy a bejövő adatok megfeleljenek a megadott szabályoknak, és hogy a felhasználó regisztrálása sikeres legyen.

Metódusok:

authorize(): bool

Leírás: Ellenőrzi, hogy a felhasználó jogosult-e a regisztrációs kérés végrehajtására.

Kimeneti adatok:

true: Mivel a metódus mindig true-t ad vissza, bármilyen felhasználó jogosult a kérés végrehajtására.

rules(): array

Leírás: Visszaadja a validációs (érvényesítési) szabályokat, amelyek érvényesek a regisztrációs űrlap benyújtására.

Kimeneti adatok:

array: A validációs szabályokat tartalmazó tömb, amely a következő mezőkre vonatkozik:

name: Kötelező, legalább 3, legfeljebb 20 karakter, és egyedi a users táblában (figyelembe véve a soft delete műveleteket).

email: Kötelező, email formátumú, és egyedi a users táblában (figyelembe véve a soft delete műveleteket).

password: Kötelező, legalább 8 karakter hosszú, és tartalmaznia kell kisbetűt, nagybetűt és számot.

confirm_password: Kötelező, a jelszó megerősítése, amelynek egyeznie kell a password mezővel.

messages(): array

Leírás: Visszaadja az egyedi hibaüzeneteket a validációs szabályokhoz, amelyeket a felhasználó láthat.

Kimeneti adatok:

array: Olyan kulcs-érték párok, ahol a kulcsok a validációs szabályok nevei, az értékek pedig a hibaüzenetek. Például:

"name.required": "Név kötelező."

"name.unique": "Létező név."

"password.regex": "A jelszónak tartalmaznia kell kisbetűt, nagybetűt és számot."

"confirm_password.same": "Nem egyező jelszó."

failedValidation(Validator \$validator)

Leírás: Kezeli a validációs hibákat, és JSON válasz formájában adja vissza azokat.

Bejövő paraméterek:

\$validator: A validációs hibák részleteit tartalmazó objektum.

Kimeneti adatok:

JSON válasz, amely tartalmazza:

"success": false

"message": "Adatbeviteli hiba"

"error": A validációs hibákat tartalmazó tömb.

LoginRequest

Feladata: A HTTP kérés érvényesítése a felhasználói bejelentkezés számára. Biztosítja, hogy a bejövő adatok megfeleljenek a megadott szabályoknak, és megfelelő válaszokat ad, ha az adatok nem érvényesek.

Metódusok:

authorize(): bool

Leírás: Ellenőrzi, hogy a felhasználó jogosult-e a kérés végrehajtására.

Kimeneti adatok:

true: Minden felhasználónak engedélyezi a kérés végrehajtását.

rules(): array

Leírás: Visszaadja a validációs (érvényesítési) szabályokat, amelyek érvényesek a bejövő kérésre.

Kimeneti adatok:

array: A validációs szabályokat tartalmazó tömb, amely a következő mezőkre vonatkozik:

name: Kötelező mező, a felhasználó neve.

password: Kötelező mező, a felhasználó jelszava.

messages(): array

Leírás: Visszaadja az egyedi hibaüzeneteket a validációs szabályokhoz, amelyeket a felhasználó láthat.

Kimeneti adatok:

array: Olyan kulcs-érték párok, ahol a kulcsok a szabályok nevei, az értékek pedig a hibaüzenetek, például:

"name.required": "Név kötelező"

"password.required": "Jelszó kötelező"

failedValidation(Validator \$validator)

Leírás: Kezeli a validációs hibákat, és JSON válasz formájában adja vissza azokat.

Bejövő paraméterek:

\$validator: A validációs hibák részleteit tartalmazó objektum.

Kimeneti adatok:

JSON válasz, amely tartalmazza:

"success": false

"message": "Adatbeviteli hiba"

"error": A validációs hibákat tartalmazó tömb.

ModifyUserRequest

Feladata: A HTTP kérés érvényesítése a felhasználói adatok módosításakor. Biztosítja, hogy a bejövő adatok megfeleljenek a megadott szabályoknak és hogy a felhasználó jogosult legyen az adatok módosítására.

Metódusok:

authorize(): bool

Leírás: Ellenőrzi, hogy a felhasználó jogosult-e a kérés végrehajtására.

Kimeneti adatok:

true: Ha a felhasználó "super" vagy "admin" jogosultsággal rendelkezik, akkor a kérés engedélyezett.

false: Ha a felhasználó nem rendelkezik ezekkel a jogosultságokkal, akkor a kérés nem engedélyezett.

rules(): array

Leírás: Visszaadja a validációs (érvényesítési) szabályokat a módosítandó felhasználói adatokhoz.

Kimeneti adatok:

array: A validációs szabályokat tartalmazó tömb, amely a következő mezőkre vonatkozik:

id: Kötelező, léteznie kell a users táblában.

name: Kötelező, minimum 3, maximum 20 karakter, egyedi a users táblában (a módosítandó id figyelembevételével).

email: Kötelező, email formátumú, egyedi a users táblában (a módosítandó id figyelembevételével).

password: Opcionális, legalább 8 karakter hosszú, és kell, hogy tartalmazzon kisbetűt, nagybetűt és számot.

(A admin mező, amelyet kommentáltál, a jogosultságok kezelésére lenne hivatott, de nincs jelenleg aktiválva).

messages(): array

Leírás: Visszaadja az egyedi hibaüzeneteket a validációs szabályokhoz, amelyeket a felhasználó láthat.

Kimeneti adatok:

array: Olyan kulcs-érték párok, ahol a kulcsok a szabályok nevei, az értékek pedig a hibaüzenetek, például:

"id.required": "Felhasználó azonosító megadása kötelező."

"email.email": "Nem megfelelő formátum."

"name.required": "Név kötelező."

(Bármely más hibaüzenet, amely az öt szerszámhoz tartozik).

failedValidation(Validator \$validator)

Leírás: Kezeli a validációs hibákat, és JSON válasz formájában adja vissza azokat.

Bejövő paraméterek:

\$validator: A validációs hibák részleteit tartalmazó objektum.

Kimeneti adatok:

JSON válasz, amely tartalmazza:

"success": false

"message": "Adatbeviteli hiba"

"error": A validációs hibákat tartalmazó tömb.

EventRequest

Feladata: A HTTP kérés érvényesítése az események létrehozásához és frissítéséhez. Biztosítja, hogy a bejövő adatok megfeleljenek a megadott szabályoknak, és megfelelő válaszokat ad, ha az adatok nem érvényesek.

Metódusok:

authorize(): bool

Leírás: Ellenőrzi, hogy a felhasználó jogosult-e a kérés végrehajtására.

Kimeneti adatok:

true: Minden felhasználónak engedélyezi a kérés végrehajtását.

rules(): array

Leírás: Visszaadja a validation (érvényesítési) szabályokat, amelyek alkalmazandók a bejövő kérésre.

Kimeneti adatok:

array: A validációs szabályokat tartalmazó tömb. A következő mezőkre vonatkozik:

name: Kötelező, minimum 2, maximum 40 karakter, egyedi a events táblában.

startDate: Dátum formátum, a endDate előtt vagy egyenlő lehet, nem kötelező.

endDate: Dátum formátum, a startDate után vagy egyenlő lehet, nem kötelező.

startTime: Időformátum, nem kötelező.

endTime: Időformátum, nem kötelező.

description: Opcionális szöveg.

locationName, locationcountry, address, state: Opcionális szövegek.

weblink: Opcionális URL formátum.

latitude, longitude: Opcionális numerikus értékek.

gpx: Opcionális szöveg.

Megjegyzés: A name mező egyedi ellenőrzése a kérés típusától függően változik: PATCH vagy PUT metódus esetén figyelembe veszi az esemény ID-ját, hogy elkerülje a saját nevekkel való ütközést.

messages(): array

Leírás: Visszaadja az egyedi hibaüzeneteket a validációs szabályokhoz, amelyek orientálják a felhasználót az érvényesítési hibák esetén.

Kimeneti adatok:

array: Olyan kulcs-érték párok, ahol a kulcsok a szabályok nevei, az értékek pedig a hibaüzenetek, például:

"name.required": "Esemény neve elvárt."

"startDate.date": "Érvényes kezdeti dátum szükséges."

(És így tovább a többi mező hibaüzenetével).

failedValidation(Validator \$validator)

Leírás: Kezeli a validációs hibákat, és JSON válasz formájában adja vissza azokat.

Bejövő paraméterek:

\$validator: A validációs hibák részleteit tartalmazó objektum.

Kimeneti adatok:

JSON válasz, amely tartalmazza:

"success": false

"message": "Adatbeviteli hiba"

"error": A validációs hibákat tartalmazó tömb.

SubscribeRequest

Feladata: A HTTP kérés érvényesítése a felhasználók eseményekhez való feliratkozásakor. Biztosítja, hogy a bejövő adatok megfeleljenek a megadott szabályoknak, és lehetővé teszi az eseményekhez való kapcsolódást.

Metódusok:

authorize()

Leírás: Ellenőrzi, hogy a felhasználó jogosult-e a feliratkozási kérés végrehajtására.

Kimeneti adatok:

true: Mivel a metódus mindig true-t ad vissza, bármilyen felhasználó jogosult a kérés végrehajtására.

rules()

Leírás: Visszaadja a validációs (érvényesítési) szabályokat, amelyek érvényesek a feliratkozási űrlap benyújtására.

Kimeneti adatok:

array: A validációs szabályokat tartalmazó tömb, amely a következő mezőkre vonatkozik:

events_id: Kötelező, léteznie kell az events táblában (ellenőrzi, hogy az esemény az adatbázisban van-e).

comment: Opcionális, ha megadva, akkor karakterlánc formátumú.

TagRequest

Feladata: A HTTP kérés érvényesítése a címkék létrehozásakor vagy frissítésekor. Biztosítja, hogy a bejövő adatok megfeleljenek a megadott szabályoknak, és hogy a címkék kezelése megfelelően történjen.

Metódusok:

authorize(): bool

Leírás: Ellenőrzi, hogy a felhasználó jogosult-e a címke létrehozási vagy frissítési kérés végrehajtására.

Kimeneti adatok:

true: Mivel a metódus mindig true-t ad vissza, bármilyen felhasználó jogosult a kérés végrehajtására.

rules(): array

Leírás: Visszaadja a validációs (érvényesítési) szabályokat, amelyek érvényesek a címke adataira.

Kimeneti adatok:

array: A validációs szabályokat tartalmazó tömb, amely a következő mezőkre vonatkozik:

name: Kötelező mező, minimum 2, maximum 20 karakter hosszúságú.

group: Kötelező mező.

Feltételes logika:

Ha a kérést PATCH vagy PUT metódussal küldik, a validációs szabályhoz hozzáadódik a unique:tags,name,\$tagId feltétel, ahol \$tagId a frissítendő címke azonosítója.

Más esetben az `unique:tags,name` szabály alkalmazandó, amely biztosítja, hogy az új címke neve egyedi legyen.

`messages()`

Leírás: Visszaadja az egyedi hibaüzeneteket a validációs szabályokhoz, amelyeket a felhasználó láthat.

Kimeneti adatok:

`array`: Olyan kulcs-érték párok, ahol a kulcsok a validációs szabályok nevei, az értékek pedig a hibaüzenetek. Például:

`"name.required"`: "Címke neve elvárt."

`"name.unique"`: "Címke már létezik ezzel a megnevezéssel."

`"name.min"`: "Túl rövid címke név."

`"name.max"`: "Túl hosszú címke név."

`"group.required"`: "Csoport elvárt."

`failedValidation(Validator $validator)`

Leírás: Kezeli a validációs hibákat, és JSON válasz formájában adja vissza azokat.

Bejövő paraméterek:

`$validator`: A validációs hibák részleteit tartalmazó objektum.

Kimeneti adatok:

JSON válasz, amely tartalmazza:

`"success"`: `false`

`"message"`: "Adatbeviteli hiba"

`"error"`: A validációs hibákat tartalmazó tömb.

Middleware

AdminAuthMiddleware

Feladata: A felhasználók admin jogosultságainak ellenőrzése a beérkező HTTP kérések során. Megakadályozza a nem jogosult felhasználók hozzáférését a védett erőforrásokhoz.

Metódusok:

`handle($request, Closure $next)`

Leírás: Ellenőrzi, hogy a felhasználó be van-e jelentkezve és rendelkezik-e admin jogosultságokkal. Ha a felhasználó nem autentikált vagy nem rendelkezik admin jogosultságokkal, a feldolgozás megszakad, és hibaüzenetet küldd vissza.

Bejövő paraméterek:

\$request: A beérkező HTTP kérés.

\$next: A következő middleware vagy a végpont kezelője.

Kimenő adatok:

Ha a felhasználó nincs autentikálva: JSON válasz, amely tartalmazza a következő üzenetet: 'User not authenticated' és a HTTP státusz kód: 401 Unauthorized.

Ha a felhasználó nem rendelkezik admin jogosultsággal: JSON válasz, amely tartalmaz egy hibaüzenetet: {"error": "Azonosítási hiba", "message": "Nincs jogosultság"} és a HTTP státusz kód: 403 Forbidden.

Ha a felhasználó autentikált és jogosult: A kérés átadása a következő middleware-nek vagy a végpont kezelőjének.

Mail

EmailVerification

Feladata: E-mail üzenetek létrehozása a felhasználói e-mail címek megerősítésére. Ez az osztály kezeli az e-mail sablonokat, az üzenet tartalmát, valamint a szükséges adatokat a megerősítő link generálásához.

Használat: Az osztály használatával egyszerűen létrehozhatunk megerősítő e-maileket, amelyet elküldhetünk az új felhasználók számára a regisztráció során.

Metódusok:

__construct(User \$user)

Leírás: Az osztály konstruktora, amely a felhasználó adatokat inicializálja.

Bejövő paraméterek:

\$user: A felhasználói információkat tartalmazó User objektum.

envelope(): Envelope

Leírás: Visszaadja az üzenet borítékát, beleértve a tárgyat.

Kimeneti adatok:

Envelope: Az üzenet borítékát reprezentáló objektum, amely tartalmazza a subject mezőt.

content(): Content

Leírás: Visszaadja az üzenet tartalmának definícióját, beleértve a sablont és az átadott változókat.

Kimeneti adatok:

Content: Az üzenet tartalmát képviselő objektum. A tartalom a view mezőben megadott emails.verify sablont használja, amelyhez hozzárendel egy verificationUrl -t.

attachments(): array

Leírás: A visszaadott tömb az üzenethez csatolt fájlokat tartalmazza. Jelen esetben nem tartalmaz csatolmányokat, üres tömböt ad vissza.

Kimeneti adatok:

array: Az üzenethez csatolt fájlok (ha léteznek) tömbje.

Provider

AppServiceProvider

Feladata: A AppServiceProvider osztály a Laravel szolgáltatási regisztrátorának és bootstrapping osztályának szerepét tölti be. Itt lehetőség van különböző alkalmazás-szolgáltatások regisztrálására, valamint az alkalmazás indításakor végrehajtandó logikák definiálására. Különösen fontos szerepet játszik a felhasználói jogosultságok kezelésében is.

Tulajdonságok és Metódusok:

public function register(): void

Leírás: Az alkalmazás szolgáltatásainak regisztrálására szolgáló metódus. Itt általában a különböző alkalmazás-specifikus szolgáltatásokat regisztrálhatjuk.

Implementáció: Jelen esetben a metódus üres, de a későbbiekben itt érkezhettek be a különböző szolgáltatások.

Jogosultság: Fejlesztői forrásként szolgál, ahol a további szolgáltatások, függőségek, vagy harmadik fél csomagok regisztrálása történhet.

```
public function boot(): void
```

Leírás: Az alkalmazás szolgáltatásainak inicializálására szolgáló metódus. Itt hajtható végre az a logika, amely végigfut az alkalmazás indításakor.

Implementáció:

Gate::before(): A Gate rendszerrel kapcsolatos jogosultságok ellenőrzéséhez használható. Az admin mező értéke 2 esetén a metódus true értéket ad vissza, így a felhasználó automatikusan minden jogosultságot megkap szuper adminként.

Gate::define('admin', ..): A admin jogosultság definiálása. Ha a felhasználó admin mezője 1, akkor a metódus true értéket ad vissza.

Adatbázis

| cache tábla | | |
|-------------|---------------|------------------|
| Mező | Típus / hossz | Leírás |
| key | varchar(255) | nincs alkalmazva |
| value | mediumtext | nincs alkalmazva |
| expiration | int(11) | nincs alkalmazva |

A cache tábla nincs alkalmazva

| cache_locks tábla | | |
|-------------------|---------------|------------------|
| Mező | Típus / hossz | Leírás |
| key | varchar(255) | nincs alkalmazva |
| owner | varchar(255) | nincs alkalmazva |
| expiration | int(11) | nincs alkalmazva |

A cache_locks tábla nincs alkalmazva

| comments tábla | | |
|----------------|----------------------------|---|
| Mező | Típus / hossz | Leírás |
| users_id | bigint(20) | users táblából id mint idegenkulcs |
| events_id | bigint(20) | events táblából id mint idegenkulcs |
| comment | text utf8mb4_unicode_ci | felhasználók által írt hozzászólás szövege |
| rating | int(11) | felhasználó saját esemény értékelése (nincs alkalmazva) |
| created_at | timestamp | létrehozás dátuma és pontos ideje |
| updated_at | timestamp | módosítás dátuma és |

| | | |
|--|--|--------------|
| | | pontos ideje |
|--|--|--------------|

A comments tábla nincs alkalmazva

| events tábla | | |
|-----------------|------------------------------------|---|
| Mező | Típus / hossz | Leírás |
| id | bigint(20) UNSIGNED | esemény egyedi azonosítója automatikusan kapja |
| name | varchar(255) utf8mb4_unicode_ci | esemény megnevezése |
| startDate | date | esemény kezdetének a dátuma |
| endDate | date | esemény végének a dátuma |
| startTime | time | esemény kezdetének ideje |
| endTime | time | esemény végének ideje |
| description | text utf8mb4_unicode_ci | esemény leírása |
| image | varchar(255) utf8mb4_unicode_ci | kép hivatkozása |
| locationName | varchar(255) utf8mb4_unicode_ci | esemény helyének a megnevezése |
| locationcountry | varchar(255) utf8mb4_unicode_ci | esemény országának a megnevezése |
| address | varchar(255) utf8mb4_unicode_ci | esemény helyének a címe |
| state | int(11) | az eseménynek az állapota (nincs alkalmazva) |
| weblink | varchar(255) utf8mb4_unicode_ci | az eseményhez tartozó weboldal linkje |

| | | |
|------------|------------------------------------|--|
| latitude | decimal(10,7) | földrajzi szélesség a gps api számára (nincs alkalmazva) |
| longitude | decimal(10,7) | földrajzi hosszúság a gps api számára (nincs alkalmazva) |
| gpx | varchar(255) utf8mb4_unicode_ci | adat a gps számára (nincs alkalmazva) |
| subscribed | int(10) | az eseményre jelenleg feliratkozott felhasználók száma |
| created_at | timestamp | létrehozás dátuma és pontos ideje |
| updated_at | timestamp | módosítás dátuma és pontos ideje |

| eventtags tábla | | |
|-----------------|---------------|--|
| Mező | Típus / hossz | Leírás |
| events_id | bigint(20) | events táblából id mint idegenkulcs |
| tags_id | bigint(20) | tags táblából id mint idegenkulcs |
| created_at | timestamp | létrehozás dátuma és pontos ideje |
| updated_at | timestamp | módosítás dátuma és pontos ideje |

| favorites tábla | | |
|-----------------|---------------|---------------------------------------|
| Mező | Típus / hossz | Leírás |
| users_id | bigint(20) | users táblából id mint idegenkulcs |
| events_id | bigint(20) | events táblából id mint |

| | | |
|------------|-----------|-----------------------------------|
| | | idegenkulcs |
| created_at | timestamp | létrehozás dátuma és pontos ideje |
| updated_at | timestamp | módosítás dátuma és pontos ideje |

A favorites tábla nincs alkalmazva

| migrations tábla | | |
|------------------|------------------------------------|---|
| Mező | Típus / hossz | Leírás |
| id | int(10) UNSIGNED | migrációs tábla egyedi azonosítója automatikusan kapja |
| migration | varchar(255) utf8mb4_unicode_ci | migrációs tábla fájl neve |
| batch | int(11) | migrációs tábla lefutásának száma |

| password_reset_tokens tábla | | |
|-----------------------------|------------------------------------|------------------|
| Mező | Típus / hossz | Leírás |
| email | varchar(255) utf8mb4_unicode_ci | nincs alkalmazva |
| token | varchar(255) utf8mb4_unicode_ci | nincs alkalmazva |
| created_at | timestamp | nincs alkalmazva |

A password_reset_tokens tábla nincs alkalmazva

| personal_access_tokens tábla | | |
|------------------------------|------------------------------------|---|
| Mező | Típus / hossz | Leírás |
| id | bigint(20) UNSIGNED | token egyedi azonosítója automatikusan kapja |
| tokenable_type | varchar(255) utf8mb4_unicode_ci | token típusa |
| tokenable_id | bigint(20) UNSIGNED | tokenhez tartozó |

| | | |
|--------------|------------------------------------|--|
| | | azonosító jelen esetben userekhez tartoznak |
| name | varchar(255) utf8mb4_unicode_ci | token megnevezése |
| token | varchar(64) utf8mb4_unicode_ci | token hashelve |
| abilities | text utf8mb4_unicode_ci | tokenben kezelt jogosultságok (nincs alkalmazva) |
| last_used_at | timestamp | token legutóbbi felhasználásának az ideje |
| expires_at | timestamp | token lejáratí ideje (nincs alkalmazva) |
| created_at | timestamp | létrehozás dátuma és pontos ideje |
| updated_at | timestamp | módosítás dátuma és pontos ideje |

| sessions tábla | | |
|----------------|------------------------------------|---|
| Mező | Típus / hossz | Leírás |
| id | varchar(255) utf8mb4_unicode_ci | munkamenethez tartozó azonosító |
| user_id | bigint(20) | munkamenethez tartozó felhasználó azonosítója |
| ip_address | varchar(255) utf8mb4_unicode_ci | munkamenet ip címe |
| user_agent | text utf8mb4_unicode_ci | munkamenet kliensalkalmazásának a megnevezése |
| payload | longtext utf8mb4_unicode_ci | töltet szerializált formában |
| last_activity | bigint(11) | unixos timestamp |

| subscriptions tábla | | |
|---------------------|----------------------------|--|
| Mező | Típus / hossz | Leírás |
| id | bigint(20) UNSIGNED | feliratkozás egyedi azonosítója automatikusan kapja |
| users_id | bigint(20) | users táblából id mint idegenkulcs |
| events_id | bigint(20) | events táblából id mint idegenkulcs |
| comment | text utf8mb4_unicode_ci | felhasználó által írt élménybeszámoló |
| sub_date | date | feliratkozás ideje |
| created_at | timestamp | létrehozás dátuma és pontos ideje |
| updated_at | timestamp | módosítás dátuma és pontos ideje |

| tags tábla | | |
|------------|------------------------------------|---|
| Mező | Típus / hossz | Leírás |
| id | bigint(20) UNSIGNED | címke egyedi azonosítója automatikusan kapja |
| name | varchar(255) utf8mb4_unicode_ci | címke megnevezése |
| group | varchar(255) utf8mb4_unicode_ci | címke besorolási kategóriája (beégetett) |
| created_at | timestamp | létrehozás dátuma és pontos ideje |
| updated_at | timestamp | módosítás dátuma és pontos ideje |

| tag_subscriptions tábla | | |
|-------------------------|---------------|------------------------------------|
| Mező | Típus / hossz | Leírás |
| users_id | bigint(20) | users táblából id mint idegenkulcs |
| tags_id | bigint(20) | tags táblából id mint idegenkulcs |
| created_at | timestamp | létrehozás dátuma és pontos ideje |
| updated_at | timestamp | módosítás dátuma és pontos ideje |

| users tábla | | |
|-------------------|------------------------------------|--|
| Mező | Típus / hossz | Leírás |
| id | bigint(20) UNSIGNED | felhasználó egyedi azonosítója automatikusan kapja |
| name | varchar(255) utf8mb4_unicode_ci | felhasználó megnevezése |
| email | varchar(255) utf8mb4_unicode_ci | felhasználó e-mail címe |
| email_verified_at | timestamp | regisztráció visszaigazolás dátuma. ha nincs visszaigazolva, akkor NULL |
| password | varchar(255) utf8mb4_unicode_ci | felhasználó jelszava letitkosítva |
| admin | int(11) | felhasználó jogosultsága |
| remember_token | varchar(255) utf8mb4_unicode_ci | nincs alkalmazva |
| created_at | timestamp | létrehozás dátuma és pontos ideje |
| updated_at | timestamp | módosítás dátuma és |

| | | |
|------------|-----------|-------------------------------|
| | | pontos ideje |
| deleted_at | timestamp | törlés dátuma és pontos ideje |