

# CommCord

Projekt menedzsment rendszer Laravelben

Csányi Gergő Zsombor

Kazai Dániel Gábor

Szamosközi Máté

# Vizsgaremek adatlap

A Vizsgaremek készítői:

Neve: Csányi Gergő Zsombor  
E-mail címe: csanyi.gergo.zsombor@blathy.info

Neve: Kazai Dániel Gábor  
E-mail címe: kazai.daniel.gabor@blathy.info

Neve: Szamosközi Máté  
E-mail címe: szamoskozi.mate@blathy.info

A Vizsgaremek témája:

A vizsgaremekünk témája egy

A Vizsgaremek címe:

Projekt menedzsment rendszer Laravelben

Kelt: Budapest, 2025. Április 30.

..Csányi Gergő Zsombor

Csányi Gergő Zsombor

Kazai Dániel Gábor

Kazai Dániel Gábor

Szamosközi Máté

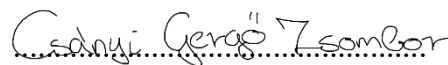
Szamosközi Máté

A Vizsgaremek készítőinek aláírása

# Eredetiség nyilatkozat

Alulírott tanuló kijelentem, hogy a vizsgaremek saját és csapattársa(i)m munkájának eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült vizsgaremekrészét képező anyagokat az intézmény archiválhatja és felhasználhatja.

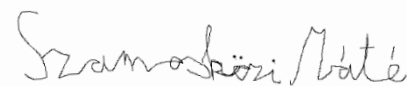
Kelt: Budapest, 2022. június.16.



Csányi Gergő  
Tanuló aláírása



Kazai Dániel Gábor  
Tanuló aláírása



Szamosközi Máté  
Tanuló aláírása

# Tartalomjegyzék

Projekt leírása .....	- 5 -
Frontend dokumentáció .....	- 6 -
Migrációk dokumentáció .....	- 11 -
Adatbázis dokumentáció.....	- 15 -
Modell dokumentáció .....	- 18 -
Kontroller dokumentáció .....	- 26 -
Kérések dokumentáció.....	- 40 -
Api Végpontok dokumentáció .....	- 51 -

# Projekt leírása

A csapatprojektünk egy játékközpontú tematikus blogoldal, amelyben a felhasználók tudnak posztokat írni bármely hosszúságban, és ezeket meg is tudják osztani a weboldalon, hogy más felhasználók lássák, olvassák, illetve kommenteljenek rá, ezzel interakciókat lehet létesíteni felhasználók között. Van lehetőség fiókok létrehozására is, amely segít abban, hogy meg lehessen állapítani, ki írta az adott megjegyzést vagy bejegyzést, ugyanis ezeket nem lehet egy meglévő fiók nélkül. A már megírt és közzétett bejegyzéseket lehet utólag törölni, illetve módosítani is, ha esetlegesen a felhasználó meg akarja változtatni a gondolatait, vagy bővíteni azokon. A projekthez hozzá lehet férni egy kifejezetten admin felhasználóval is, aki bármely posztot, bejegyzést vagy profilt törölhet, módosíthat. Ez segítené egy élesben futó rendszernek a moderálását és megfelelő működését.

## -- FELHASZNÁLÓI DOKUMENTÁCIÓ --

A főoldalon található navigációs sáv segítségével a felhasználó könnyen navigálhat az alábbi funkciók között: Főoldal (Home), Új téma létrehozása (Create Threads), Témák kezelése (Threads Manager), Felhasználói profil (User Profile), valamint a bejelentkezési/regisztrációs felület (Login/Register). A jobb felső sarokban található téma váltó gomb segítségével egy kattintással válthatunk világos és sötét nézet között.

A regisztráció során a felhasználó megadja nevét, e-mail címét és jelszavát. Ezután bejelentkezéskor a rendszer hitelesíti az adatokat, és sikeres azonosítás után a felhasználó hozzáfér az összes elérhető funkcióhoz. A hitelesített állapotot localStorage tárolja, így újratöltés után is fennmarad a bejelentkezés.

A felhasználói profil oldalon megtekinthetők a bejelentkezett felhasználó adatai (név, email, szerepkör). Innen lehet kijelentkezni is. Az admin szerepkör külön jogosultságokat biztosít, például mások hozzászólásainak szerkesztését is.

Az új téma létrehozása oldalon a felhasználó megadhatja a bejegyzés címét, tartalmát és kiválaszthatja a kategóriát. A sikeresen létrehozott szálak megjelennek a rendszeren belül, más felhasználók által is elérhetők. Minden poszt mellé hozzászólás is írható.

A Threads Manager nézetben az összes elérhető bejegyzés listázva van bal oldalt. Egy bejegyzésre kattintva megjelenik annak tartalma és a hozzászólások. A hozzászólások alatt található egy űrlap új komment írásához. A saját vagy admin által írt hozzászólások szerkeszthetők és törölhetők is. A kommentekhez tartozó időbélyeg is megjelenik.

A felhasználói élmény fokozására a projekt támogatja a sötét és világos témát, amely a teljes oldal megjelenésére hatással van, beleértve a sidebar-t, komment háttereket és szövegszíneket.

Az oldal technológiai háttere Vue 3 keretrendszerre és Vite építőrendszerre épül. A komponensek között egyedi nézetek (views) és komponensek biztosítják az újrafelhasználhatóságot. Az adatok REST API-n keresztül kerülnek betöltésre. A jogosultságkezelés localStorage alapon történik.

Ez az alkalmazás ideális egyszerű fórumrendszer kiépítésére, amely fejleszthető például hitelesítés middleware-rel, CSRF védelemmel vagy komplexebb jogosultságkezeléssel.

## -- FELJESZTŐI DOKUMENTÁCIÓ --

### Munkamegosztás

A csapat 3 főből áll, minden egyes csapattag képes volt a projekt bizonyos részeihez (backend-frontend) hozzátenni. A munka az alkalmazás adatbázis és backend kezelésével kezdődött, amely során egyenlően el volt osztva a munka. A controllerek, seederek, migrálások el voltak osztva, viszont a modellek írása már egy emberes munka volt Szamosközi Máté részéről. Az alkalmazás frontend része a prototípus megalkotása óta szintén együttes munka volt, a részletesebb kivitelezés és végső javítások és továbbfejlesztések Csányi Gergő által voltak megcsinálva. A fejlesztés közben felmerült problémák és hibák közös megoldására nem mindig volt lehetőség magánügyi dolgok és események révén, emiatt ezen dolgok Kazai Dániel által voltak orvosolva az esetek nagyobb részén. Ezek végett el lehet mondani, hogy a fejlesztés alatti munkamegosztás sikeres volt, minden csapattag tudott írni minden részéhez a programnak, és teljes a csapatmunka sikeressége.

## Frontend dokumentáció

**- src/:** Az src/ könyvtár a frontend fejlesztés központi része. Ez az a hely, ahol minden vizuális elem és interakció megszületik, karbantartásra kerül és dinamikusan összekapcsolódik az alkalmazás más részeivel. Akár új oldalt szeretnénk hozzáadni, akár egy meglévő gomb működését módosítani, mindig itt kell keresni a kódot. A jó felépítésű src/ mappa átláthatóságot, skálázhatóságot és hatékony fejlesztést biztosít minden csapat vagy egyéni fejlesztő számára. Ez a (source = forrás) könyvtár a Vue-alapú frontend alkalmazás szíve-lelke. Ez a mappa tartalmazza az alkalmazás működéséhez szükséges teljes forráskódot, azaz a Vue-komponenseket, a nézeteket (views), a belépési pontként szolgáló fájlokat, valamint minden vizuális és logikai elemet, amit a felhasználó az oldalon lát és használ.

Ez az a hely, ahol a fejlesztők ténylegesen dolgoznak: új funkciókat fejlesztenek, meglévő funkciókat módosítanak, vagy hibákat javítanak. Bármilyen módosítás, ami az oldal működését vagy kinézetét befolyásolja, itt történik.

**- public/:** Ez a mappa minden statikus fájlt (pl. képek, favicon, nyers HTML fájlok) tartalmaz, amelyek nem mennek keresztül a Vite build folyamatán. Ezek közvetlenül elérhetők az alkalmazásban is.

**- .vscode/:** Fejlesztői környezethez tartozó konfigurációs mappa, pl. formázási szabályokat, beállított bővítményeket tartalmazhat, de nem szükséges a működéshez, inkább a fejlesztői élményt segíti.

**- node\_modules/:** Automatikusan generált könyvtár, amely a package.json alapján telepített összes függőséget tárolja. Itt található a Vue.js, Vite, axios és egyéb könyvtárak kódja. Nem ajánlott szerkeszteni benne bármit is. Az npm csomagok telepítése során települ le ez is.

- **.gitignore**: Meghatározza, mely fájlokat és mappákat ne kövessen a Git verziókezelő rendszere. Például a node\_modules, vagy .env fájlok ilyenek.

- **index.html**: A belépési pont HTML szinten. Ez az egyetlen HTML fájl, amit a böngésző valójában betölt. Innen indul a Vue app, itt van például a <div id="app">, amit a main.js kód betölt és vezérel. Ez csak egy alapot ad, amit átszerkeszt minden nézet, és komponens az App.vue fájlban keresztül.

- **jsconfig.json**: Fejlesztői fájl, ami IntelliSense-t, aliasolást és útvonalkezelést segít VSCode-ban vagy más fejlesztői környezetben. Elősegíti a modulimportok átláthatóságát.

- **package.json**: Az egyik legfontosabb konfigurációs fájl, ami tartalmazza az összes függőséget, scripteket(pl. npm run dev), verziószámokat, a projektünk nevét, és sok más alap adatot.

- **package-lock.json**: A package.json pontosabb, zárolt változata. Rögzíti, hogy pontosan melyik verziójú csomagokat használjuk. Fontos a konzisztens fejlesztői környezethez.

**views/ mappa**: Ebben a mappában találhatóak az úgynevezett "nézetek" (views), melyek konkrét oldalakat képviselnek az alkalmazásban. Ilyen például:

- HomeView.vue: a főoldal
- LoginRegisterView.vue: a bejelentkezés és regisztráció felülete
- PostsView.vue: a hozzászólások/posztok kezelése
- UserProfile.vue: a felhasználói fiók adatainak megjelenítése
- CreateThreadView.vue: új bejegyzés létrehozása

- **README.md**: Markdown-formátumban írt dokumentációs fájl, amelyet a GitHub automatikusan megjelenít. Általában tartalmazza a projekt leírását, indítási lépéseket, fejlesztési instrukciókat, vagy bármit amit a fejlesztő érdemesnek tart beleírni.

- **vite.config.js**: A Vite build tool konfigurációs fájlja. Itt lehet módosítani például, globális CSS fájlokat, proxy beállításokat API hívásokhoz. Nagyon fontos, ha minél egyedibb felépítést szeretnél.

- **index.js** Vue Router konfigurációját tartalmazza, amely felelős az alkalmazás nézetek (views) közötti navigációjáért. A fájlban először importáljuk a szükséges elemeket: a createRouter és createWebHistory függvényeket a vue-router csomagból, majd az alkalmazásban használt nézeteket (HomeView, CreateThreadView, LoginRegisterView, UserProfile, PostsView). Ezután definiálunk egy routes nevű tömböt, amely minden oldalhoz hozzárendel egy útvonalat (path), egy nevet (name), és a megjelenítendő Vue-komponenst (component). Például a főoldal (/) esetében a HomeView komponenst rendeljük hozzá. Hasonlóan megadjuk az útvonalakat az új szálak létrehozásához (/create-threads), a bejelentkezéshez és regisztrációhoz (/login), a felhasználói profilhoz (/profile), valamint a posztok listázásához (/posts). Végül létrehozunk egy router példányt a createRouter hívással, amely a createWebHistory függvényt használja az

URL-ek kezelésére (hash nélkül, tiszta URL-ek). Az így létrehozott routert alapértelmezettként exportáljuk, így az egész alkalmazásban használható lesz.

## App.vue

**Elhelyezkedés:** Projekt gyökér (src/)

**Funkció:** Teljes frontend alkalmazás váza, globális elrendezés és téma-kezelés.

Az App.vue az egész alkalmazás központi váza, amely meghatározza az oldalstruktúrát és vezérli a sötét/világos téma váltását. A sablon (<template>) szekcióban található egy teljes navigációs menü (navbar), amely router-linkek segítségével különböző nézetek (Home, Create Threads, Threads Manager, User Profile, Login/Register) között biztosít navigációt. A navigáció két részre van osztva: bal oldali (nav-left) és jobb oldali (nav-right) linkekre, ami reszponzív, modern felépítést tesz lehetővé.

A téma váltás funkció a theme-switch elemen keresztül történik, ahol egy checkbox váltja ki a világos és sötét mód közötti váltást. Ez Vue v-model segítségével köti össze a data() részben található isDark változóval. A toggleTheme() függvény dinamikusan módosítja a body elem osztályát (dark), és elmenti az aktuális beállítást a localStorage-be ('theme' kulccsal). A mounted() hook ellenőrzi az induláskor elmentett témát, és automatikusan alkalmazza azt. Ez biztosítja, hogy a felhasználó következő látogatáskor ugyanabban a témában folytathatja a böngészést.

A <router-view /> komponens helyet biztosít a dinamikusan változó aloldalak megjelenítésére a Vue Router segítségével. Ez lehetővé teszi, hogy minden oldal csak a tartalom részét töltsse be, a navigációs sáv változatlanul megmaradjon az oldalon belüli navigációk során.

A stílusok között szerepel globális alapbeállítások (body, html { margin: 0; padding: 0; }) mellett a dark osztály definíciója is, amely sötét háttérszínt (#121212) és világos szöveget (#fff) állít be. A navbar zöld színű (rgb(1, 119, 17)), rugalmasan igazítva a két oldalra. A navigációs linkek (nav-link) fehérek, aktív állapotban félkövérek (bold) és hover eseménynél aláhúzásos effekt jelenik meg.

A téma kapcsoló (theme-switch) egy stílusosan lekerekített csúszka (slider) animációval, amely modern megjelenést ad a funkciónak. A kapcsoló bejelölt állapotában zöld színű (#4caf50) háttér jelenik meg.

Összességében az App.vue a projekt szíve: itt történik az alap struktúra felépítése, a témaváltás kezelése, a navigációs sáv megjelenítése, és a felhasználói élmény alapvető szabályozása.

## HomeView.vue

**Elhelyezkedés:** src/views/

**Funkció:** Az alkalmazás kezdőoldala, üdvözlő szöveg és rövid bemutatkozás megjelenítése.

Ez a Vue komponens egy egyszerű felépítésű kezdőoldalt valósít meg. A HomeView.vue semmilyen adatot nem kér az API-tól, nem tartalmaz reaktív változókat (ref vagy



reactive), és nem használ külső modulokat vagy komplex logikát. A célja, hogy az alkalmazásba belépő felhasználók számára egy statikus, barátságos üdvözlést nyújtson.

A sablonban (<template>) egy központosított div elem található, amely egy főcímet (h1) és egy bekezdést (p) jelenít meg. Ez az oldal tartalmazza az eddigi létrehozott bejegyzéseket, így a felhasználó láthatja mind.

A style scoped szekció biztosítja, hogy az oldal megjelenése különbözzön a többi oldaltól. Az itteni stílus a tartalom középre helyezését helyezi előtérbe, alap szövegszín és háttérszín beállítása, amely figyelembe veszi az alkalmazás aktuális témáját (sötét vagy világos mód). A színek és a háttér dinamikusan alkalmazkodnak a body osztályához, így támogatva a világos/sötét módot.

Összességében a HomeView.vue egy minimalista, funkcionális belépőoldal, amely kiválóan alkalmas arra, hogy a felhasználók áttekintést kapjanak a platform működéséről, mielőtt a főbb funkciókat elkezdnék használni.

## CreateThreadView.vue

**Elhelyezkedés:** src/views/

**Funkció:** Új fórumtémák (thread-ek) létrehozása felhasználói felületen keresztül.

Ez a Vue komponens egy űrlap-alapú nézetet valósít meg, amely lehetővé teszi a bejelentkezett felhasználók számára új fórumtémák létrehozását, valamint az előre meghatározott témák kiválasztását. A komponens ref típusú reaktív változókat használ az új poszt címének, tartalmának és kiválasztott kategóriájának tárolására. A createPost() metódus egy POST kérést küld az API-nak (/api/posts), amely létrehozza az új bejegyzést az adatbázisban, ha be van jelentkezve a felhasználó, és minden adata a bejegyzésnek ki van töltve.

Ezen kívül a komponens tartalmaz némi örökölt logikát is a kommentek kezelésére, valamint bejegyzések és hozzászólások megjelenítésére. A stílusok egy része osztály-alapú, scoped formában van megvalósítva, biztosítva a CSS izolációt. Az űrlap mezők validációja egyszerű, főként required attribútumokra épül.

A nézet kinyeri az adatbázisból, a localStorage-ba a user\_id értékét, ezáltal hitelesített felhasználóként posztol a bejelentkezett felhasználó. A kategória választás select mező biztosítja, ahol a kategóriák lokálisan vannak deklarálva egy tömbben.

## PostsView.vue

**Elhelyezkedés:** src/views/

**Funkció:** Fórum jellegű bejegyzések (threadek) és azok kommentjeinek megtekintése, valamint hozzászóláskezelés (CRUD).

Ez a komponens egy teljes értékű „fórum thread kezelő” felület, ahol a bal oldali sávban (sidebar) listázódnak a létező bejegyzések (postok), kattintásra pedig megjelenik a teljes bejegyzés tartalma, a hozzátartozó kommentekkel együtt. A komponens fetchPosts és fetchCommentsForSelectedPost függvényeken keresztül kommunikál a Laravel alapú backenddel, GET típusú API-hívásokat használva.

A comments tömb minden posthoz szűrt adatokkal töltődik fel, ahol a user\_id, content, és created\_at mezők kerülnek megjelenítésre. A submitNewComment metódus POST kérést küld új hozzászólások létrehozásához, míg az edit, delete és update funkciók a hozzászólások módosítását teszik lehetővé – mindezt jogosultságalapú logika szerint. A jogosultság ellenőrzését a canEditOrDelete() függvény végzi a localStorage-ből kiolvasott felhasználói azonosító és szerepkör alapján.

Stílus szempontjából a komponens teljesen reszponzív, a sötét/világos témákhoz igazodik a body.dark vagy body:not(.dark) CSS osztályok alapján. A felhasználói élményt timestamp és interaktív szerkesztési felület javítja, így a PostsView.vue az alkalmazás egyik legösszetettebb, ugyanakkor legfontosabb vizuális és funkcionális egysége.

## LoginRegisterView.vue

**Elhelyezkedés:** src/views/

**Funkció:** Felhasználói regisztráció és bejelentkezés kezelése egy közös nézeten belül.

Ez a komponens egy teljes körű felhasználókezelési felületet biztosít, ahol a látogatók regisztrálhatnak egy új fiókot, majd bejelentkezhetnek. A Vue komponens két külön űrlapot tartalmaz: az egyik új felhasználók regisztrációjára, a másik meglévő felhasználók bejelentkezésére szolgál. Mindkét űrlap v-model kötésekkel használja, hogy a felhasználó által beírt adatokat a ref-ekben tárolja: registerName, registerEmail, registerPassword, loginEmail, loginPassword.

A komponens két aszinkron függvényt definiál: register és login. Mindkettő előtt előzetesen lekérésre kerül a csrf-cookie, majd fetch-en keresztül HTTP POST kérés történik a megfelelő Laravel API végpontokra (/api/register, /api/login). Sikeres bejelentkezés esetén a felhasználó adatai localStorage-be kerülnek, és átirányítás történik a dashboardra.

(ez a megoldás a jövőben tovább fejlesztésre szorul)

A komponens reagál a message változóra, amivel visszajelzést ad a felhasználónak az eseményekről. A dizájn egy mobilbarát, letisztult formát követ, középre rendezett elrendezéssel.

Ez a nézet kulcsszerepet játszik az alkalmazás hitelesítési logikájában, és közvetlen kapcsolatban áll a Laravel backenddel.

## UserProfile.vue

**Elhelyezkedés:** src/views/

**Funkció:** Felhasználói profil megjelenítése, bejelentkezett adatok kezelése és kijelentkezés biztosítása.

Ez a komponens felelős a felhasználó profiloldalának megjelenítéséért. A user nevű reaktív változó a localStorage-ből tölti be a felhasználó nevét, email címét és szerepkör azonosítóját (role\_id) a komponens onMounted() hook-jában.

A vizuális megjelenést a card nevű div elem biztosítja: egy avatar kép (avatar szolgáltatásról alap avatarral), név, email cím és szerepkör ("Admin" vagy "User") információk jelennek meg. A kijelentkezés gomb (Logout) törli a localStorage-ban tárolt felhasználói adatokat, majd átirányítja a felhasználót a bejelentkező oldalra.

A designban reszponzív, középre igazított kártyastílus (card) van alkalmazva, árnyékkhatással és lekerekített sarkokkal. A sötét és világos téma támogatása a CSS változók (var(--bg-color), var(--text-color)) segítségével történik. A logout gomb külön kiemelve élénk piros árnyalattal jelenik meg, amely hover eseménnyel még sötétebbé válik a vizuális visszajelzés érdekében.

Összességében a UserProfile.vue egyszerű, mégis felhasználóbarát módon biztosítja a profiladatok megtekintését és a biztonságos kijelentkezést.

## Migrációk dokumentáció

A migráció egy olyan folyamat, amely során a megadott névvel és mezőkkel létrehoz a keretrendszer egy ilyen nevű táblát ilyen mezőkkel a projekthez kötött adatbázisban. Ezt lehet úgy is, hogy a táblák között vannak kapcsolatok is.

### Create\_categories\_table

Ezzel a migrációval a categories nevű táblát hozzuk létre, amely a posztokat látja el egy, vagy több kategóriával, ezzel könnyebbé téve a kategória szerinti rendszerezést a blogok között. A táblában kettő mező van: id és category\_name. Az id alapján lehet kapcsolni a category\_name elemeket a posztokhoz, ezzel is az id egy elsődleges kulcs.

### Create\_roles\_table

Ezzel a migrációval a roles nevű táblát hozzuk létre, amely a felhasználókat látja el egy és csak is egyetlen szereppel, amely lehet vagy admin, vagy user, így szűrni lehet a regisztrált felhasználók hatáskörét és szerepeit. A táblában kettő mező van: id és

role\_name. Az id alapján lehet kapcsolni a role\_name elemeket a felhasználókhoz, ezzel is az id egy elsődleges kulcs.

## Create\_users\_table

Ezzel a migrációval a users nevű táblát hozzuk létre, amely a felhasználások tárolásáért felel, akár új regisztrálás van, akár korábbi fiókról beszélünk. Ebben a táblában 5 mező van, amelyek a következők: id (primary key), name (string), role\_id (int, foreign key), email (string, unique), és password (string). Az id felel a felhasználó azonosítására id alapján, ezzel ez az elsődleges kulcs is. A name az a regisztrált fióknak megadott név. A role\_id az másodlagos kulcs, amin keresztül össze van kapcsolva a tábla a roles táblával, ami arra szolgál, hogy a felhasználóknak meg lehessen adni, milyen szerepük és hatáskörük van (vagy admin, vagy sima user). Az email a regisztrált fióknak megadott email cím, ami kötelezően egyedi minden esetben. Ez azt jelenti és eredményezi, hogy egy megadott email cím nem tartozhat egynél több felhasználóhoz. A password a regisztrált fióknak megadott jelszó, amely tokenizált titkosítással van bevédeve.

## Create\_comments\_table

Ezzel a migrációval a comments nevű táblát hozzuk létre, amely a posztok alatt, felhasználók által hagyott kommentek gyűjtésére és tárolására szolgál, akár új, akár régebbi kommentekről beszélünk. Ebben a táblában négy mező van, amelyek: id (primary key), post\_id (int, foreign key), user\_id (int, foreign key), content (string). Az id a komment azonosításáért felel, ez az elsődleges kulcs. A post\_id másodlagos kulcs, amely segítségével össze van kapcsolva a posts táblával. Ez funkcionalitásban azt jelenti, hogy azok a kommentek, amelyek össze vannak kapcsolva az adott bejegyzésekkel, csak az adott posztok alatt jelennek meg. A user\_id másodlagos kulcs, amely segítségével felhasználóhoz lehet kötni a kommenteket. Ezzel azt érjük el, hogy lehessen látni, melyik felhasználó írta az adott hozzászólást. A content a kommenteknek a tartalmának tárolásáért felelnek.

## Create\_posts\_table

Ezzel a migrációval a posts nevű táblát hozzuk létre, amelyek a felhasználók által közzétett bejegyzések tárolására szolgálnak. A táblában öt mező van: id (primary key), user\_id (int, foreign key), category\_id (int, foreign key), title (string), body (string). Az id a bejegyzés azonosításáért felel, amellyel a többi táblában post\_id-ként utalunk, ezáltal ez egy elsődleges kulcs. A user\_id másodlagos kulcs, amelynek segítségével felhasználóhoz lehet kötni a bejegyzéseket. A category\_id másodlagos kulcs, amelynek segítségével kategóriákat lehet sorolni a bejegyzésekhez. A title a bejegyzések címének tárolásáért felel. A body a bejegyzések törzsének/tartalmának tárolásáért felel.

### Adatbázis seeder fájlok

A seeder fájlok az adatbázisba való alap adatok feltöltésére szolgálnak. Ezeket lehet megadni több különböző módon; egyesével lehet megadni adatokat, emellett többesével

is. Van mód arra is, hogy ezt egy – a Laravelbe beépített - pluginnal, a Fakerrel lehessen megcsinálni. Ehhez adat gyárat (vagyis factorykat) kell hasznosítani, így megadott hosszúságú karaktersorokat, mondatokat, szavakat, és sok más adatot lehet véletlenszerűen feltölteni. Modelleket is lehet használni, amelyek reprezentálják a táblákat a Laravel keretrendszerben, a könnyebb futásért.

## CommentsSeeder

Ez a fájl a comments tábla adatokkal való feltöltésére van használva. Ehhez nekünk szükségünk van a kettő másodlagos kulcshoz tartozó azonosító adatokhoz (post\_id és user\_id), amelyeket a másik kettő táblából kiszedünk a pluck függvényt használva, ezeket meg a nekik létrehozott tömbben tároljuk a későbbi feltöltéshez. A tábla feltöltésekor a user\_id és post\_id véletlenszerűen lesznek feltöltve a korábban említett tömbök használatával az array\_rand függvény segítségével. A content az a fake függvény által van feltöltve, amely text formátummal tölt fel adatokat, maximum megadva ötven karakter hosszúsággal. Ez a feltöltési folyamat egy for ciklusban van megírva, így lehet irányítani, hogy mennyi adatsor kerüljön fel a táblába. Ebben a fájlban mi tíz darab sort töltünk fel a comments táblába.

## CategoriesSeeder

Ez a fájl a categories tábla adatokkal való feltöltésére van használva. Ehhez nekünk annyit kellett tennünk, hogy egy tömböt létrehoztunk, amibe adatokat egyesével tettünk. Majd ezen tömb alapján töltjük fel a táblát annyi sorral, amennyi kategóriát előre megadtunk a tömbben egy for ciklus segítségével.

## PostsSeeder

Ez a fájl a posts tábla adatokkal való feltöltésére van használva. Ehhez nekünk szükségünk van a kettő másodlagos kulcshoz tartozó azonosító adatokhoz (user\_id és category\_id), amelyeket a másik kettő táblából kiszedünk a pluck függvényt használva, ezeket meg a nekik létrehozott tömbben tároljuk a későbbi feltöltéshez. A tábla feltöltésekor a user\_id és category\_id véletlenszerűen lesznek feltöltve a korábban említett tömbök használatával az array\_rand függvény segítségével. A title és a body az a fake függvény által van feltöltve, amelyek közül az előbbi a sentence, az utóbbi a text formátummal tölt fel adatokat, maximum megadva hat és ötven karakter hosszúsággal egyaránt. Ez a feltöltési folyamat egy for ciklusban van megírva, így lehet irányítani, hogy mennyi adatsor kerüljön fel a táblába. Ebben a fájlban mi tíz darab sort töltünk fel a posts táblába.

## RolesSeeder

Ez a seeder fájl a roles tábla adatokkal való feltöltésére szolgál. Mi ehhez megadtunk egy roles tömböt, amelyben kettő adatot tárolunk, nevesen user és admin, amelyeket mi feltöltünk egy for ciklus segítségével a táblába.

## UserSeeder

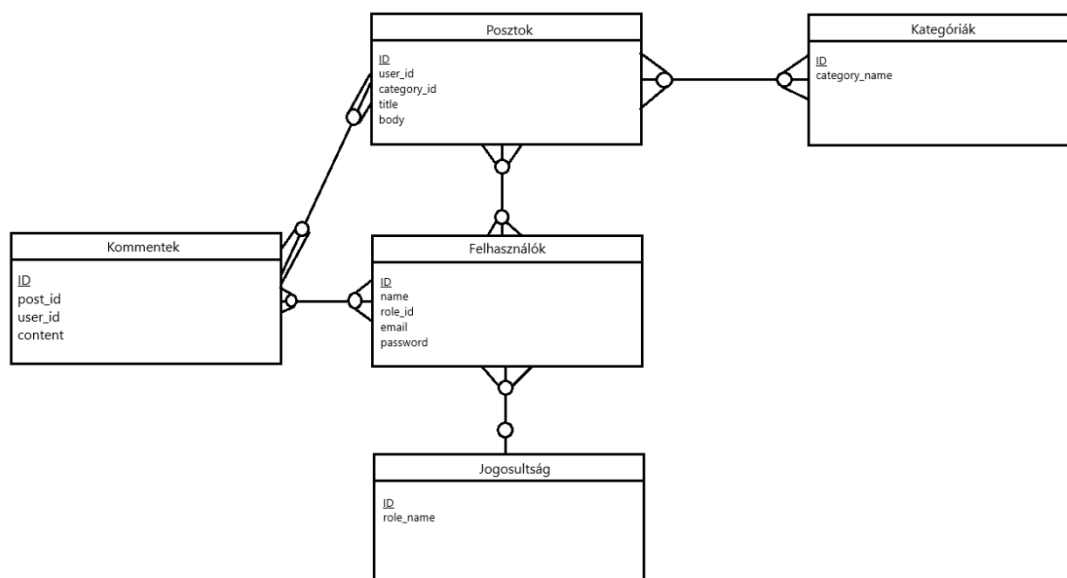
Ez a fájl a users tábla adatokkal való feltöltésére van használva. Ehhez nekünk szükségünk van az egy darab másodlagos kulcshoz tartozó azonosító adatokhoz (role\_id), amelyet a másik táblából kisorszunk a pluck függvényt használva, ezeket az adatokat meg a nekik létrehozott tömbben tároljuk a későbbi feltöltéshez. A tábla feltöltésekor a role\_id véletlenszerűen lesz feltöltve a korábban említett tömb használatával az array\_rand függvény segítségével. A name és az email az a fake függvény által van feltöltve, amelyek közül az előbbi a text, az utóbbi az email formátummal tölt fel adatokat, maximum megadva hat és húsz karakter hosszúsággal egyaránt. A password a név adatát használva lesz feltöltve, amelyhez a titkosítás miatt egy hashelő algoritmuson keresztül lesz átalakítva a Hash osztályhoz tartozó make függvény segítségével. Ez a feltöltési folyamat egy for ciklusban van megírva, így lehet irányítani, hogy mennyi adatsor kerüljön fel a táblába. Ebben a fájlban mi húsz darab sort töltünk fel a users táblába.

## DatabaseSeeder

Ez az alap – a Laravel keretrendszer által generált - seeder fájl, amit mi arra használunk, hogy a többi seedert meghívjuk a call függvényt használva, így mindegyik seedert le lehet futtatni egy seeder fájl futtatásával.

# Adatbázis dokumentáció

## Diagram és leírása



Ezen az EK (elem-kapcsolat) diagramon lehet látni a projekt során felhasznált adattáblákat, azon elemeit és kapcsolatait. Az EK diagram arra szolgál, hogy az egyes elemek és adatok között milyen kapcsolatok állnak fenn, hogyan viszonyulnak azok egymáshoz, így reprezentálva a táblák közötti kapcsolatot és az adatok egymáshoz fűződő viszonyokat és azok logikáit.

## Mezők részletes ismertetése

Mindegyik mező fontos a program és projekt futtatása és azon adatai kezelése során.

A posztok azok az oldal fő elemei, amelyekben a felhasználók szabadon írhatnak a kategóriának megfelelő bejegyzéseket. Az ide tartozó tábla a következőket tartalmazza: id, user\_id, category\_id, title, body. Az id arra felel, hogy mindegyik posztnak legyen egy olyan adata, amellyel be lehet azokat azonosítani a többi táblával és adattal való megfelelő kapcsolathoz. A user\_id arra felel, hogy össze legyen kapcsolva a felhasználó táblához. Funkcionalitásban ez azt jelenti, hogy lehessen látni, melyik posztot melyik felhasználó írta. A category\_id arra szolgál, hogy össze legyen kapcsolva a kategória táblához. Ez a gyakorlatban arra van, hogy azonosító alapján lehessen kategóriákat kötni bejegyzésekhez. A title egy bejegyzés címéért, a body meg a bejegyzésnek a tartalmáért felel, illetve azon adatok tárolására.

A kategória táblában a posztokhoz tartozó kategóriák tárolására és azonosítására van használva. A táblában csak 2 adat van tárolva. Ezek az id és a category\_name. Minden kategória névhez tartozik egy darab egyedi azonosító szám, amely alapján lehet utalni arra a kategóriára.

A kommentek tábla a bejegyzések alatt megjelenő kommentek tárolásáért felel. A táblában 4 adat van eltárolva, ezek az adatok a következők: id, post\_id, user\_id és content. Az azonosító, mint a többi táblában is, arra van, hogy a kommenteket be lehessen azonosítani ez alapján. A post\_id arra van, hogy a kommenteket csak az adott azonosítóval ellátott bejegyzéshez lehessen kötni és csak annál a posztnál látszódjának használat alatt. A user\_id hasonló szerepet lát el, viszont ez a megadott azonosítóval rendelkező felhasználóhoz legyen kötve a megadott komment. A content pedig a komment tartalmának tárolására felel.

A felhasználók tábla az oldalon regisztrált és használt fiókok tárolására van. A tábla elemei: id, name, role\_id, email, password. Az id a felhasználók egyedi szám szerinti azonosítására szolgál, amelyet más táblákban is használunk és kapcsolunk, hogy az adott elemeket lehessen hozzákötni a felhasználókhoz. A name a felhasználó nevének tárolására szolgál, ez lehet bármi, viszont csak a felhasználók elkülönítésére szolgál, belépéskor ez nincs használva. A role\_id a jogosultsági besorolás megállapítására van, amely vagy admin lehet, vagy user. Az email a regisztráció során megadott email cím, amelyet használni kell a bejelentkezés során. Viszont ezen adat minden adatsorban (vagyis felhasználónként) kötelezően különbözik, amely megegyezik sok életbeli programmal és rendszerrel is. A password pedig a felhasználó által megadott jelszó, amely kell a bejelentkezésnél. Ezen adat a Laravel keretrendszernek köszönhetően titkosítási algoritmuson keresztül le van titkosítva, így tokenizáltan kerül feltöltésre, ezáltal esetleges biztonsági kockázatok csökkentve vannak.

A jogosultság tábla az oldalon regisztrált felhasználók jogosultságának tárolására szolgál. A tábla elemei: id, role\_name. A táblában kettő adatsor van, amelyeknek külön azonosító száma van, illetve megkülönböztetésre kerül kettő jogosultság és szerepkör; admin és user. Az admin képes az összes adatot manipulálni a weboldalon, a user viszont csak a maga által feltöltött adatot tudja változtatni. Minden felhasználó profil automatikusan adminként van kezelve, ez csak a nem publikus buildnél lesz ez maradandó. A továbbfejlesztés során ez még meg lesz változtatva.

## Kapcsolatok indoklása, ismertetése

Az adatbázisban használt táblák közötti kapcsolatok mindegyike fontos az oldal megfelelő és rendeltetés szerű működéséhez. A felhasználók táblában és a posztok táblában van a legtöbb kapcsolat (három-három egyaránt), ez mutatja a kettő tábla és adathalmaz fontosságát is a projektben.

A felhasználók tábla a posztok, a kommentek és a jogosultság táblákhoz kapcsolódik, amely táblákban a felhasználó a user\_id másodlagos kulccsal van összekapcsolva a felhasználók táblán belüli id mezővel. Ennek fontossága magyarázható azzal, hogy a posztok és kommentek mind egy adott felhasználóhoz vannak kötve, így meg lehet állapítani, az adott közzétett tartalmak mely felhasználóhoz tartozik. A jogosultság táblához kapcsolatot azzal lehet megvédeni, hogy az egyes regisztrált felhasználók között különbséget lehet és kell is tennünk a jogosultságai és szerepkörei között. A mi esetünkben ez kettő szerepben nyilvánul meg; lehet egy felhasználó vagy admin, vagy user. Az előbbi képes az oldal bármely részén manipulálni adatot, az utóbbi meg csak az adott userhez tartozó feltöltött tartalmat tudja irányítani és változtatni. Ennek lényege, hogy a felületet fejlesztő és ellenőrző csapat vagy ember tudja felügyelni, milyen tartalmat osztanak meg, és ha olyat talál valaki, amely nem felel meg esetlegesen adott iránnyelvekkel, vagy pedig szimplán sértő és jogellenes tartalom került fel az oldalra.



Ilyen esetben az admin jogosultsággal ellátott profil szabadon tudja változtatni, vagy esetlegesen törölni az adott tartalmat; legyen az akár bejegyzés, vagy pedig komment.

Következő kapcsolatok, amelyek elengedhetetlen fontossággal rendelkeznek, az a posztok táblának a kapcsolatai a felhasználó, a kommentek és a kategóriák táblákhoz. Az első esetben a `user_id` által van kapcsolva a felhasználók táblában lévő id-hez. A második esetben a `comment_id` alapján kapcsolódik a kommentek táblában található id-hez. Az utolsó esetben meg a `category_id` alapján a kategóriák tábla id mezőjéhez. A felhasználók-posztok kapcsolatot a továbbiakban említve lettek, viszont ugyanúgy meg lehet említeni a fontosságát. Célja a posztok felhasználóhoz való rendelése, így megállapíthatóvá válna, melyik poszt melyik felhasználóhoz tartozik. A kommentek pedig egyszerűen elmagyarázható; mint minden online blogos, fórumos oldalon, itt is lehet kommenteket tenni a bejegyzések alá, ezzel hozzá lehet szólni és további beszélgetéseket folytatni. A kategóriák kapcsolata pedig azzal magyarázható, hogy minden poszt egy bizonyos témát pedzeget, amely szerint sokkal könnyebben lehet keresni és szűrni, ha van kategóriák közötti megkülönböztetés. Ezt lehetővé teszi a kategóriák tábla és a posztok tábla közötti kapcsolat.

## Továbbfejlesztési lehetőségek:

Ami az adatbázist és adatkezelést illeti, továbbfejlesztési lehetőségek közé tartozik a képek esetleges integrálása. Mind saját, felhasználó által feltölthető képek használata, mind a szerverre alaphoz feltöltött képek tárolása egyaránt hasznos lehet a program további életszakaszaiban. Ezen képek használata lehet akár a bejegyzések, kommentek közé, valamint a profilképek beállítására is hasznos lehet. Továbbá az adatbázist lehet bővíteni olyan mezőkkel, amelyek segítik a kapcsolatok további fenntartását felhasználók között. Esetleg olyan táblával is, amely megoldást nyújthat egy rendes beszélgetőfelülethez, amely csak az adott felhasználókat érintik. Emellett lehet olyan lehetőség is a későbbiekben, amely azt engedné, hogy egy felhasználó által megadott poszt vagy komment csak az ő általa engedélyezettek emberek által legyen csak látható egy online barát és szociális menü kontextusában. Ezek mellett a szociális interakciók továbbá fejlesztésénél lehet adott felhasználókat tiltani is a nem kívánatos kapcsolatok és kommunikációk elkerülésére. Ezekhez mind külön táblák és mezők, illetve további kapcsolatok is szükségesek, de mind kivitelezhető a megfelelő körülmények között és a legmegfelelőbb módokon.

A `LocalStorage`-ban tárolni adatokat sajnos nem a legmegbízhatóbb megoldás, hiszen bárki akinek van egy kevés hozzáértése Javascripthoz, le tudja kérdezni az éppen bejelentkezett felhasználó adatait, ami nem biztonságos a jövőre nézve. Optimalizáció céljából hasznos lenne, ha például a `Navbar` egy komponensként lenne létrehozva. Jelenleg is tökéletesen ellátja a célját, viszont nem lehet a jövőben is felhasználni másik projektben. Nem egy verzatilis megoldás jelenleg. Érdemes lenne globális állapotkezelőt bevezetni, hogy ne kelljen minden komponensben külön lekérni a felhasználói adatokat, hanem központilag kezelhető legyen a belépett állapot és a felhasználói szerepkör. A validációk jelenleg nagyrészt kliensoldalon történnek, de hosszabb távon fontos lenne a hibakezelést (pl. üzenet küldés sikertelensége, hálózati hibák) egységes és felhasználóbarát módon megoldani, akár modális figyelmeztetésekkel. Ugyanígy a kommentek vagy bejegyzések listájának frissítése is

lehetne dinamikusabb, például websockets vagy polling használatával. További fejlesztési lehetőségként szóba jöhet a profilkép feltöltésének lehetősége, jelszóváltoztatás, hozzászólások szűrése kategória vagy népszerűség szerint, illetve egy admin dashboard statisztikákkal.

## Modell dokumentáció

### Osztály: User

A User modell a users adatbázistáblát reprezentálja, amely a rendszerben regisztrált felhasználók adatait tárolja. Ez a modell tartalmazza a kitölthető, rejtett és típuskonvertált attribútumokat, valamint a kapcsolódásokat más modellekhez, például a Posts, Comments és Roles modellekhez Attribútumok:

#### 1. \$fillable Leírás:

- a. -A tömegesen kitölthető attribútumokat határozza meg. Ez a funkció megkönnyíti az adatok kezelését, miközben védi a modellt a nem kívánt mezők módosításától.
- b. Attribútumok:
  - i. -name: A felhasználó neve.
  - ii. -role\_id: Azonosító, amely a felhasználó szerepkörére mutat.
  - iii. -email: A felhasználó e-mail címe. -password: A felhasználó jelszava (titkosított).
  - iv. Kód részlet:

```
<?php
protected $fillable = [
    'name',
    'role_id',
    'email',
    'password',
];
```

#### 2. \$hidden Leírás:

- a. -A sorosítás során rejtett attribútumokat határozza meg. Ezek az attribútumok nem jelennek meg, amikor a modellt JSON vagy más formátumba alakítjuk. Ez különösen hasznos az érzékeny adatok, például a jelszó elrejtésére.
- b. Attribútumok:
  - i. -password: A felhasználó jelszava.
  - ii. -remember\_token: A "remember me" token.protected function casts(): Meghatározza az attribútumokat, amelyeket specifikus típusokra kell konvertálni.

iii. Kód részlet:

```
<?php
protected $hidden = [
    'password',
    'remember_token',
];
```

3. \$casts Leírás:

- a. -Az attribútumok automatikus típuskonvertálását biztosítja. Ezek az attribútumok automatikusan átalakulnak a megadott típusra, amikor a modellt betöltjük vagy mentjük.
- b. Attribútumok:
  - i. -email\_verified\_at: Dátum/idő formátumra konvertálva.
  - ii. -password: Titkosított formátumra konvertálva.
  - iii. Kód részlet:

```
<?php
protected function casts(): array
{
    return [
        'email_verified_at' => 'datetime',
        'password' => 'hashed',
    ];
}
```

## Kapcsolatok:

1. posts() Leírás:

- a. -Ez a kapcsolat azt jelzi, hogy egy felhasználó több bejegyzést is létrehozhat. A hasMany kapcsolatot használja, amely az Eloquent ORM-ben a "több van hozzá kapcsolva" kapcsolatot jelenti.
- b. Kapcsolat típusa:
  - i. -hasMany

Kód részlet:

```
<?php
public function posts(){
    return $this->hasMany(Posts::class);
}
```

2. comments() Leírás:

- a. -Ez a kapcsolat azt jelzi, hogy egy felhasználó több kommentet is írhat. A hasMany kapcsolatot használja.
- b. Kapcsolat típusa:
  - i. -hasMany

Kód részlet:

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

3. roles() Leírás:

- a. -Ez a kapcsolat azt jelzi, hogy egy felhasználó több szerepkörrel is rendelkezhet. A hasMany kapcsolatot használja.
- b. Kapcsolat típusa:
  - i. -hasMany

Kód részlet:

```
<?php
public function roles(){
    return $this->hasMany(Roles::class);
}
```

## Osztály: Categories

A Categories modell a categories adatbázistáblát reprezentálja, amely a rendszerben elérhető kategóriákat tárolja. Ez a modell tartalmazza a kitölthető attribútumokat, valamint a kapcsolódásokat más modellekhez, például a Posts modellhez.

### Attribútumok:

1. \$fillable

Leírás:

-A tömegesen kitölthető attribútumokat határozza meg. Ez a funkció megkönnyíti az adatok kezelését, miközben védi a modellt a nem kívánt mezők módosításától.

## Attribútumok:

-category\_name: A kategória neve.

Kód részlet:

```
<?php
protected $fillable = [
    'category_name'
];
```

## Kapcsolatok:

1. posts()

Leírás:

-Ez a kapcsolat azt jelzi, hogy egy kategória, egy vagy több bejegyzéshez tartozhat. A belongsTo kapcsolatot használja, amely az Eloquent ORM-ben a "tartozik valamihez" kapcsolatot jelenti.

Kapcsolat típusa:

-belongsTo

Kód részlet:

```
<?php
public function posts(){
    return $this->belongsTo(Posts::class);
}
```

## Osztály: Comments

A Comments modell a comments adatbázistáblát reprezentálja, amely a rendszerben létrehozott kommenteket tárolja. Ez a modell tartalmazza a kitölthető attribútumokat, valamint a kapcsolódásokat más modellekhez, például a User és Posts modellekhez.

## Attribútumok:

1. \$fillable

Leírás:

-A tömegesen kitölthető attribútumokat határozza meg. Ez a funkció megkönnyíti az adatok kezelését, miközben védi a modellt a nem kívánt mezők módosításától.

## Attribútumok:

- user\_id: Azonosító, amely a kommentet létrehozó felhasználóra mutat.
- post\_id: Azonosító, amely a kommenthez tartozó bejegyzésre mutat.
- title: A komment címe.
- body: A komment szövege.

Kód részlet:

```
<?php
protected $fillable = [
    'user_id',
    'post_id',
    'title',
    'body',
];
```

## Kapcsolatok:

1. users()

Leírás:

-Ez a kapcsolat azt jelzi, hogy egy komment egyetlen felhasználóhoz tartozik. A belongsTo kapcsolatot használja, amely az Eloquent ORM-ben a "tartozik valakihez" kapcsolatot jelenti.

## Kapcsolat típusa:

- belongsTo

Kód részlet:

```
<?php
public function users(){
    return $this->belongsTo(User::class);
}
```

2. posts()

Leírás:

-Ez a kapcsolat azt jelzi, hogy egy komment egyetlen bejegyzéshez tartozik. A belongsTo kapcsolatot használja.

### Kapcsolat típusa:

-belongsTo

Kód részlet:

```
<?php
public function posts(){
    return $this->belongsTo(Posts::class);
}
```

### Osztály: Posts

A Posts modell a posts adatbázistáblát reprezentálja, amely a rendszerben létrehozott bejegyzéseket tárolja. Ez a modell tartalmazza a kitölthető attribútumokat, valamint a kapcsolódásokat más modellekhez, például a User és Comments modellekhez

#### Attribútumok:

1. \$fillable

Leírás:

-A tömegesen kitölthető attribútumokat határozza meg. Ez a funkció megkönnyíti az adatok kezelését, miközben védi a modellt a nem kívánt mezők módosításától.

#### Attribútumok:

- user\_id: Azonosító, amely a bejegyzést létrehozó felhasználóra mutat.
- category\_id: Azonosító, amely a bejegyzés kategóriájára mutat.
- title: A bejegyzés címe.
- body: A bejegyzés tartalma.

Kód részlet:

```
<?php
protected $fillable = [
    'user_id',
    'category_id',
    'title',
    'body',
];
```

## Kapcsolatok:

### 1. users()

Leírás:

-Ez a kapcsolat azt jelzi, hogy egy bejegyzés egyetlen felhasználóhoz tartozik. A belongsTo kapcsolatot használja, amely az Eloquent ORM-ben a "tartozik valakihez" kapcsolatot jelenti.

Kapcsolat típusa:

-belongsTo

Kód részlet:

```
<?php
public function users(){
    return $this->belongsTo(User::class);
}
```

### 2. comments()

Leírás:

-Ez a kapcsolat azt jelzi, hogy egy bejegyzéshez több komment is tartozhat. A hasMany kapcsolatot használja, amely az Eloquent ORM-ben a "több van hozzá kapcsolva" kapcsolatot jelenti.

Kapcsolat típusa:

-hasMany

Kód részlet:

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

## Osztály: Roles

Az Roles modell a roles adatbázistáblát reprezentálja, amely a rendszerben elérhető szerepköröket tárolja. Ez a modell tartalmazza a kitölthető attribútumokat, valamint a kapcsolódásokat más modellekhez, például a User modellhez.



## Attribútumok:

### 1. \$fillable

Leírás:

-A tömegesen kitölthető attribútumokat határozza meg. Ez a funkció megkönnyíti az adatok kezelését, miközben védi a modellt a nem kívánt mezők módosításától.

## Attribútumok:

-role\_name: A szerepkör neve.

Kód részlet:

```
<?php
protected $fillable = [
    'role_name'
];
```

## Kapcsolatok:

### 1. users()

Leírás:

-Ez a kapcsolat azt jelzi, hogy egy szerepkör egy vagy több felhasználóhoz tartozhat. A belongsTo kapcsolatot használja, amely az Eloquent ORM-ben a "tartozik valakihez" kapcsolatot jelenti.

## Kapcsolat típusa:

-belongsTo

Kód részlet:

```
<?php
public function users(){
    return $this->belongsTo(User::class);
}
```

# Kontroller dokumentáció

## Osztály: AuthController:

Az AuthController felelős a felhasználók regisztrációjáért, bejelentkezéséért és kijelentkezéséért. Az osztály a Laravel keretrendszerben készült, és a hitelesítési folyamatokat kezeli.

Metódusok:

register(Request \$request)

Leírás:

-Ez a metódus új felhasználók regisztrációját kezeli. Ellenőrzi a bemeneti adatokat, titkosítja a jelszót, és elmenti az adatokat az adatbázisba.

Paraméterek:

-Request \$request: A regisztrációs kérés adatait tartalmazza.

Validációs szabályok:

-name: Kötelező, szöveg, maximum 255 karakter.

-email: Kötelező, e-mail formátum, maximum 255 karakter, egyedi.

-password: Kötelező, szöveg, minimum 6 karakter.

Válaszok:

-201 (Created): Sikeres regisztráció esetén.

-500 (Internal Server Error): Hiba esetén.

Kód részlet:

```
<?php
$validated = $request->validate([
    'name' => 'required|string|max:255',
    'email' => 'required|email|max:255|unique:users',
    'password' => 'required|string|min:6',
]);

DB::table('users')->insert([
    'name' => $validated['name'],
    'email' => $validated['email'],
    'password' => Hash::make($validated['password']),
    'role_id' => 2,
    'created_at' => now(),
    'updated_at' => now(),
]);
```

## login(Request \$request)

Leírás:

-Ez a metódus a felhasználók bejelentkezését kezeli. Ellenőrzi a hitelesítési adatokat, és ha helyesek, visszaadja a felhasználói adatokat.

Paraméterek:

-Request \$request: A bejelentkezési kérés adatait tartalmazza.

Validációs szabályok:

-email: Kötelező, e-mail formátum.

-password: Kötelező, szöveg.

Válaszok:

-200 (OK): Sikeres bejelentkezés esetén.

-401 (Unauthorized): Hibás hitelesítési adatok esetén.

Kód részlet:

```
<?php
$validated = $request->validate([
    'email' => 'required|email',
    'password' => 'required|string',
]);

if (!Auth::attempt($validated)) {
    return response()->json([
        'message' => 'Invalid credentials',
    ], 401);
}

$user = Auth::user();

return response()->json([
    'message' => 'Login successful',
    'user' => $user,
]);
```

## logout()

Leírás:

-Ez a metódus a felhasználók kijelentkezését kezeli. Érvényteleníti a munkamenetet és regenerálja a tokeneket.

Paraméterek:

-Request \$request: A kijelentkezési kérés adatait tartalmazza.

Válaszok:

-200 (OK): Sikeres kijelentkezés esetén.

Kód részlet:

```
<?php
Auth::logout();
$request->session()->invalidate();
$request->session()->regenerateToken();

return response()->json([
    'message' => 'Logged out successfully',
]);
```

## Osztály: UsersController

A UsersController osztály a felhasználók kezeléséért felelős. Ez az osztály a Laravel keretrendszerben készült, és a RESTful API szabvány szerint valósítja meg a CRUD műveleteket a User modellel kapcsolatban.

Metódusok:

index()

Leírás:

-Ez a metódus az összes felhasználó listáját adja vissza JSON formátumban.

Visszatérési érték:

-200 (OK): Sikeres lekérdezés esetén.

Kód részlet:

```
<?php
public function index()
{
    return \Response::json(
        [
            "data"=> [
                ["users"=> User::all()],
            ],
            "success"=> true,
            "message"=> "",
        ], 200
    );
}
```

## store()

Leírás:

-Ez a metódus új felhasználót hoz létre a megadott adatok alapján. A bemeneti adatokat a StoreUsersRequest validálja.

Paraméterek:

-StoreUsersRequest \$request: A felhasználó létrehozásához szükséges adatok.

Visszatérési érték:

-201 (Created): Sikeres létrehozás esetén.

Kód részlet:

```
<?php
public function store(StoreUsersRequest $request)
{
    $users = User::create($request->all());
    return response()->json($users, 201);
}
```

## update()

Leírás:

-Ez a metódus egy meglévő felhasználó adatait frissíti a megadott adatok alapján. A bemeneti adatokat az UpdateUsersRequest validálja.

Paraméterek:

-UpdateUsersRequest \$request: A felhasználó frissítéséhez szükséges adatok.

-User \$id: Az adott felhasználó példánya.

Visszatérési érték:

-201 (Created): Sikeres frissítés esetén.

Kód részlet:

```
<?php
public function update(UpdateUsersRequest $request, User $id)
{
    $id->update($request->all());
    return response()->json($id, 201);
}
```

## destroy()

Leírás:

-Ez a metódus egy meglévő felhasználót töröl az adatbázisból.

Paraméterek:

-User \$id: Az adott felhasználó példánya.

Visszatérési érték:

-200 (OK): Sikeres törlés esetén.

Kód részlet:

```
<?php
public function destroy(User $id)
{
    $id->delete();
    return response()->json($id);
}
```

## Osztály: CategoriesController

A CategoriesController osztály a kategóriák kezeléséért felelős. Ez az osztály a Laravel keretrendszerben készült, és a RESTful API szabvány szerint valósítja meg a CRUD műveleteket a Categories modellel kapcsolatban.

Metódusok:

index()

Leírás:

-Ez a metódus az összes kategória listáját adja vissza JSON formátumban.

Visszatérési érték:

-200 (OK): Sikeres lekérdezés esetén.

Kód részlet:

```
<?php
public function index()
{
    return \Response::json(
        [
            "data"=> [
                ["categories"=> Categories::all()],
            ],
            "success"=> true,
            "message"=> "",
        ], 200
    );
}
```

## store()

Leírás:

-Ez a metódus új kategóriát hoz létre a megadott adatok alapján. A bemeneti adatokat a StoreCategoriesRequest validálja.

Paraméterek:

-StoreCategoriesRequest \$request: A kategória létrehozásához szükséges adatok.

Visszatérési érték:

-201 (Created): Sikeres létrehozás esetén.

Kód részlet:

```
<?php
public function store(StoreCategoriesRequest $request)
{
    $categories = Comments::create($request->all());
    return response()->json($categories, 201);
}
```

## update()

Leírás:

-Ez a metódus egy meglévő kategória adatait frissíti a megadott adatok alapján. A bemeneti adatokat az UpdateCategoriesRequest validálja.

Paraméterek:

-UpdateCategoriesRequest \$request: A kategória frissítéséhez szükséges adatok.

-Categories \$id: Az adott kategória példánya.

Visszatérési érték:

-201 (Created): Sikeres frissítés esetén.

Kód részlet:

```
<?php
public function update(UpdateCategoriesRequest $request, Categories $id)
{
    $id->update($request->all());
    return response()->json($id, 201);
}
```

## destroy()

Leírás:

-Ez a metódus egy meglévő kategóriát töröl az adatbázisból.

Paraméterek:

-Categories \$id: Az adott kategória példánya.

Visszatérési érték:

-200 (OK): Sikeres törlés esetén.

Kód részlet:

```
<?php
public function destroy(Categories $id)
{
    $id->delete();
    return response()->json($id);
}
```

## Osztály: CommentsController

A CommentsController osztály a kommentek kezeléséért felelős. Ez az osztály a Laravel keretrendszerben készült, és a RESTful API szabvány szerint valósítja meg a CRUD műveleteket a Comments modellel kapcsolatban.

Metódusok:

### index()

Leírás:

-Ez a metódus az összes komment adatainak listáját adja vissza JSON formátumban.

Visszatérési érték:

-200 (OK): Sikeres lekérdezés esetén.



Kód részlet:

```
<?php
public function index()
{
    $comments = Comments::with('users:id,name')->get(); // eager-load user name

    $transformed = $comments->map(function ($comment) {
        return [
            'id' => $comment->id,
            'post_id' => $comment->post_id,
            'user_id' => $comment->user_id,
            'content' => $comment->content,
            'user_name' => $comment->users->name ?? 'Unknown',
        ];
    });

    return response()->json([
        'data' => $transformed
    ], 200);
}
```

## store()

Leírás:

-Ez a metódus új kommentet hoz létre a megadott adatok alapján. A bemeneti adatokat a StoreCommentsRequest validálja.

Paraméterek:

-StoreCommentsRequest \$request: A komment létrehozásához szükséges adatok.

Visszatérési érték:

-201 (Created): Sikeres létrehozás esetén.

Kód részlet:

```

<?php
public function store(StoreCommentsRequest $request)
{
    $validated = $request->validated();

    $comment = Comments::create([
        'post_id' => $validated['post_id'],
        'user_id' => $validated['user_id'],
        'content' => $validated['content'],
    ]);

    return response()->json([
        'message' => 'Comment created successfully!',
        'data' => $comment,
    ], 201);
}

```

## update()

Leírás:

-Ez a metódus egy meglévő komment adatait frissíti a megadott adatok alapján. A bemeneti adatokat az UpdateCommentsRequest validálja.

Paraméterek:

- UpdateCommentsRequest \$request: A komment frissítéséhez szükséges adatok.
- Comments \$id: Az adott komment példánya.

Visszatérési érték:

- 201 (Created): Sikeres frissítés esetén.

Kód részlet:

```

<?php
public function update(Request $request, $id)
{
    $comment = Comments::findOrFail($id);

    $comment->update([
        'content' => $request->input('content'),
    ]);

    return response()->json(['message' => 'Comment updated successfully'], 200);
}

```

## destroy()

Leírás:

-Ez a metódus egy meglévő kommentet töröl az adatbázisból.

Paraméterek:

-Comments \$id: Az adott komment példánya.

Visszatérési érték:

-200 (OK): Sikeres törlés esetén.

Kód részlet:

```
<?php
public function destroy($id)
{
    $comment = Comments::findOrFail($id);
    $comment->delete();
    return response()->json(['message' => 'Comment deleted successfully'], 200);
}
```

## Osztály: PostsController

A PostsController osztály a bejegyzések kezeléséért felelős. Ez az osztály a Laravel keretrendszerben készült, és a RESTful API szabvány szerint valósítja meg a CRUD műveleteket a Posts modellel kapcsolatban.

Metódusok:

### index()

Leírás:

-Ez a metódus az összes bejegyzés listáját adja vissza JSON formátumban.

Visszatérési érték:

-200 (OK): Sikeres lekérdezés esetén.

Kód részlet:

```
<?php
public function index()
{
    return \Response::json(
        [
            "data"=> [
                ["posts"=> Posts::all()],
            ],
            "success"=> true,
            "message"=> "",
        ], 200
    );
}
```

## store()

Leírás:

-Ez a metódus új bejegyzést hoz létre a megadott adatok alapján. A bemeneti adatokat validálja.

Paraméterek:

-Request \$request: A bejegyzés létrehozásához szükséges adatok.

Validációs szabályok:

-title: Kötelező, szöveg, maximum 255 karakter.

-body: Kötelező, szöveg.

-user\_id: Kötelező, létező felhasználói azonosító.

-category\_id: Kötelező, létező kategória azonosító.

Visszatérési érték:

-201 (Created): Sikeres létrehozás esetén.

Kód részlet:

```
<?php
public function store(Request $request)
{
    $validated = $request->validate([
        'title' => 'required|string|max:255',
        'body' => 'required|string',
        'user_id' => 'required|exists:users,id',
        'category_id' => 'required|exists:categories,id',
    ]);

    DB::table('posts')->insert([
        'title' => $request->title,
        'body' => $request->body,
        'user_id' => $request->user_id,
        'category_id' => $request->category_id,
        'created_at' => now(),
        'updated_at' => now(),
    ]);

    return response()->json([
        'message' => 'Post created successfully!'
    ], 201);
}
```

## update()

Leírás:

-Ez a metódus egy meglévő bejegyzés adatait frissíti a megadott adatok alapján. A bemeneti adatokat az UpdatePostsRequest validálja.

Paraméterek:

-UpdatePostsRequest \$request: A bejegyzés frissítéséhez szükséges adatok.

-Posts \$id: Az adott bejegyzés példánya.

Visszatérési érték:

-201 (Created): Sikeres frissítés esetén.

Kód részlet:

```
<?php
public function update(UpdatePostsRequest $request, Posts $id)
{
    $id->update($request->all());
    return response()->json($id, 201);
}
```

## destroy()

Leírás:

-Ez a metódus egy meglévő bejegyzést töröl az adatbázisból.

Paraméterek:

-Posts \$id: Az adott bejegyzés példánya.

Visszatérési érték:

-200 (OK): Sikeres törlés esetén.

Kód részlet:

```
<?php
public function destroy(Posts $id)
{
    $id->delete();
    return response()->json($id);
}
```

## Osztály: RolesController

A RolesController osztály a szerepkörök kezeléséért felelős. Ez az osztály a Laravel keretrendszerben készült, és a RESTful API szabvány szerint valósítja meg a CRUD műveleteket a Roles modellel kapcsolatban.

Metódusok:

### index()

Leírás:

-Ez a metódus az összes szerepkör listáját adja vissza JSON formátumban.

Visszatérési érték:

-200 (OK): Sikeres lekérdezés esetén.

Kód részlet:

```
<?php
public function index()
{
    return \Response::json(
        [
            "data"=> [
                ["roles"=> Roles::all()],
            ],
            "success"=> true,
            "message"=> "",
        ], 200
    );
}
```

## store()

Leírás:

-Ez a metódus új szerepkört hoz létre a megadott adatok alapján. A bemeneti adatokat a StoreRolesRequest validálja.

Paraméterek:

-StoreRolesRequest \$request: A szerepkör létrehozásához szükséges adatok.

Visszatérési érték:

-201 (Created): Sikeres létrehozás esetén.

Kód részlet:

```
<?php
public function store(StoreRolesRequest $request)
{
    $roles = Roles::create($request->all());
    return response()->json($roles, 201);
}
```

## update()

Leírás:

-Ez a metódus egy meglévő szerepkör adatait frissíti a megadott adatok alapján. A bemeneti adatokat az UpdateRolesRequest validálja.

Paraméterek:

-UpdateRolesRequest \$request: A szerepkör frissítéséhez szükséges adatok.

-Roles \$id: Az adott szerepkör példánya.

Visszatérési érték:

-201 (Created): Sikeres frissítés esetén.

Kód részlet:

```
<?php
public function update(UpdateRolesRequest $request, Roles $id)
{
    $id->update($request->all());
    return response()->json($id, 201);
}
```

## destroy()

Leírás:

-Ez a metódus egy meglévő szerepkört töröl az adatbázisból.

Paraméterek:

-Roles \$id: Az adott szerepkör példánya.

Visszatérési érték:

-200 (OK): Sikeres törlés esetén.

Kód részlet:

```
<?php
public function destroy(Roles $id)
{
    $id->delete();
    return response()->json($id);
}
```

# Kérések dokumentáció

## Osztály: StoreUsersRequest

A StoreUsersRequest osztály a felhasználók létrehozásához szükséges HTTP-kérések validálásáért felelős. Ez az osztály a Laravel keretrendszer FormRequest osztályát örökli, és lehetővé teszi a bemeneti adatok validálását és az engedélyek kezelését.

## Methods

### authorize()

Leírás:

-Ez a metódus határozza meg, hogy a felhasználó jogosult-e a kérés végrehajtására. Jelenleg minden felhasználó számára engedélyezett (true).



Visszatérési érték:

-bool: true, ha a kérés engedélyezett.

Kód részlet:

```
<?php
public function authorize(): bool
{
    return true;
}
```

## rules()

Leírás:

-Ez a metódus adja vissza a kéréshez tartozó validációs szabályokat. A felhasználó létrehozásához a következő szabályok érvényesek:

-name: Kötelező mező, amelynek értéke szöveg típusú kell legyen.

-email: Kötelező mező, amelynek értéke szöveg típusú kell legyen.

-password: Kötelező mező, amelynek értéke szöveg típusú kell legyen.

Visszatérési érték:

-array: A validációs szabályokat tartalmazó asszociatív tömb.

Validációs szabályok:

-name: Kötelező (required), szöveg (string).

-email: Kötelező (required), szöveg (string).

-password: Kötelező (required), szöveg (string).

Kód részlet:

```
<?php
public function rules(): array
{
    return [
        'name' => 'required|string',
        'email' => 'required|string',
        'password' => 'required|string'
    ];
}
```

## Osztály: StoreCategoriesRequest

A StoreCategoriesRequest osztály a kategóriák létrehozásához szükséges HTTP-kérések validálásáért felelős. Ez az osztály a Laravel keretrendszer FormRequest osztályát örökli, és lehetővé teszi a bemeneti adatok validálását és az engedélyek kezelését.

### Metódusok:

#### authorize()

Leírás:

-Ez a metódus határozza meg, hogy a felhasználó jogosult-e a kérés végrehajtására. Jelenleg minden felhasználó számára engedélyezett (true).

Visszatérési érték:

-bool: true, ha a kérés engedélyezett.

Kód részlet:

```
<?php
public function authorize(): bool
{
    return true;
}
```

#### rules()

Leírás:

-Ez a metódus adja vissza a kéréshez tartozó validációs szabályokat. A kategória létrehozásához a következő szabályok érvényesek:

-category\_name: Kötelező mező, amelynek értéke szöveg típusú kell legyen.

Visszatérési érték:

-array: A validációs szabályokat tartalmazó asszociatív tömb.

Validációs szabályok:

-category\_name: Kötelező (required), szöveg (string).

Kód részlet:

```
<?php
public function rules(): array
{
    return [
        'category_name' => 'required|string'
    ];
}
```

## Osztály: StoreCommentsRequest

A StoreCommentsRequest osztály a kommentek létrehozásához szükséges HTTP-kérések validálásáért felelős. Ez az osztály a Laravel keretrendszer FormRequest osztályát örökli, és lehetővé teszi a bemeneti adatok validálását és az engedélyek kezelését.

### Metódusok:

#### authorize()

Leírás:

-Ez a metódus határozza meg, hogy a felhasználó jogosult-e a kérés végrehajtására. Jelenleg minden felhasználó számára engedélyezett (true).

Visszatérési érték:

-bool: true, ha a kérés engedélyezett.

Kód részlet:

```
<?php
public function authorize(): bool
{
    return true;
}
```

#### rules()

Leírás:

-Ez a metódus adja vissza a kéréshez tartozó validációs szabályokat. A komment létrehozásához a következő szabályok érvényesek:

-post\_id: Kötelező mező, amelynek értéke egy létező bejegyzés azonosítója kell legyen.

-user\_id: Kötelező mező, amelynek értéke egy létező felhasználó azonosítója kell legyen.

-content: Kötelező mező, amelynek értéke szöveg típusú kell legyen, és legfeljebb 500 karakter hosszú lehet.

Visszatérési érték:

-array: A validációs szabályokat tartalmazó asszociatív tömb.

Validációs szabályok:

-post\_id: Kötelező (required), létező bejegyzés azonosítója (exists:posts,id).

-user\_id: Kötelező (required), létező felhasználó azonosítója (exists:users,id).

-content: Kötelező (required), szöveg (string), maximum 500 karakter hosszú (max:500).

Kód részlet:

```
<?php
public function rules(): array
{
    return [
        'post_id' => 'required|exists:posts,id',
        'user_id' => 'required|exists:users,id',
        'content' => 'required|string|max:500',
    ];
}
```

## Osztály: StorePostsRequest

A StorePostsRequest osztály a bejegyzések létrehozásához szükséges HTTP-kérések validálásáért felelős. Ez az osztály a Laravel keretrendszer FormRequest osztályát örökli, és lehetővé teszi a bemeneti adatok validálását és az engedélyek kezelését.

### Metódusok:

#### authorize()

Leírás:

-Ez a metódus határozza meg, hogy a felhasználó jogosult-e a kérés végrehajtására. Jelenleg minden felhasználó számára engedélyezett (true).

Visszatérési érték:

-bool: true, ha a kérés engedélyezett.

Kód részlet:

```
<?php
public function authorize(): bool
{
    return true;
}
```

#### rules()

Leírás:

-Ez a metódus adja vissza a kéréshez tartozó validációs szabályokat. A bejegyzés létrehozásához a következő szabályok érvényesek:

-title: Kötelező mező, amelynek értéke szöveg típusú kell legyen.

-body: Kötelező mező, amelynek értéke szöveg típusú kell legyen.

Visszatérési érték:

-array: A validációs szabályokat tartalmazó asszociatív tömb.

Validációs szabályok:

-title: Kötelező (required), szöveg (string).

-body: Kötelező (required), szöveg (string).

Kód részlet:

```
<?php
public function rules(): array
{
    return [
        'title' => 'required|string',
        'body' => 'required|string'
    ];
}
```

## Osztály: StoreRolesRequest

A StoreRolesRequest osztály a szerepkörök létrehozásához szükséges HTTP-kérések validálásáért felelős. Ez az osztály a Laravel keretrendszer FormRequest osztályát örökli, és lehetővé teszi a bemeneti adatok validálását és az engedélyek kezelését..

Metódusok:

### authorize()

Leírás:

-Ez a metódus határozza meg, hogy a felhasználó jogosult-e a kérés végrehajtására. Jelenleg minden felhasználó számára engedélyezett (true).

Visszatérési érték:

-bool: true, ha a kérés engedélyezett.

Kód részlet:

```
<?php
public function authorize(): bool
{
    return true;
}
```

### rules()

Leírás:

-Ez a metódus adja vissza a kéréshez tartozó validációs szabályokat. A szerepkör létrehozásához a következő szabályok érvényesek:

-role\_name: Kötelező mező, amelynek értéke szöveg típusú kell legyen.

Visszatérési érték:

-array: A validációs szabályokat tartalmazó asszociatív tömb.

Validációs szabályok:

-role\_name: Kötelező (required), szöveg (string).

Kód részlet:

```
<?php
public function rules(): array
{
    return [
        'role_name' => 'required|string'
    ];
}
```

## Osztály: UpdateUsersRequest

Az UpdateUsersRequest osztály a felhasználók frissítéséhez szükséges HTTP-kérések validálásáért felelős. Ez az osztály a Laravel keretrendszer FormRequest osztályát örökli, és lehetővé teszi a bemeneti adatok validálását és az engedélyek kezelését.

### Metódusok

#### authorize()

Leírás:

-Ez a metódus határozza meg, hogy a felhasználó jogosult-e a kérés végrehajtására. Jelenleg minden felhasználó számára engedélyezett (true).

Visszatérési érték:

-bool: true, ha a kérés engedélyezett.

Kód részlet:

```
<?php
public function authorize(): bool
{
    return true;
}
```

#### rules()

Leírás:

-Ez a metódus adja vissza a kéréshez tartozó validációs szabályokat. A felhasználó frissítéséhez a következő szabályok érvényesek:

-name: Kötelező mező, amelynek értéke szöveg típusú kell legyen.

-email: Kötelező mező, amelynek értéke szöveg típusú kell legyen.

-password: Kötelező mező, amelynek értéke szöveg típusú kell legyen.

Visszatérési érték:

-array: A validációs szabályokat tartalmazó asszociatív tömb.

Validációs szabályok:

-name: Kötelező (required), szöveg (string).

-email: Kötelező (required), szöveg (string).

-password: Kötelező (required), szöveg (string).

Kód részlet:

```
<?php
public function rules(): array
{
    return [
        'name' => 'required|string',
        'email' => 'required|string',
        'password' => 'required|string'
    ];
}
```

## Osztály: UpdateCategoriesRequest

Az UpdateCategoriesRequest osztály a kategóriák frissítéséhez szükséges HTTP-kérések validálásáért felelős. Ez az osztály a Laravel keretrendszer FormRequest osztályát örökli, és lehetővé teszi a bemeneti adatok validálását és az engedélyek kezelését.

### Metódusok

#### authorize()

Leírás:

-Ez a metódus határozza meg, hogy a felhasználó jogosult-e a kérés végrehajtására. Jelenleg minden felhasználó számára engedélyezett (true).

Visszatérési érték:

-bool: true, ha a kérés engedélyezett.

Kód részlet:

```
<?php
public function authorize(): bool
{
    return true;
}
```

## rules()

Leírás:

-Ez a metódus adja vissza a kéréshez tartozó validációs szabályokat. A kategória frissítéséhez a következő szabályok érvényesek:

-category\_name: Kötelező mező, amelynek értéke szöveg típusú kell legyen.

Visszatérési érték:

-array: A validációs szabályokat tartalmazó asszociatív tömb.

Validációs szabályok:

-category\_name: Kötelező (required), szöveg (string).

Kód részlet:

```
<?php
public function rules(): array
{
    return [
        'category_name' => 'required|string'
    ];
}
```

## Osztály: UpdateCommentsRequest

Az UpdateCommentsRequest osztály a kommentek frissítéséhez szükséges HTTP-kérések validálásáért felelős. Ez az osztály a Laravel keretrendszer FormRequest osztályát örökli, és lehetővé teszi a bemeneti adatok validálását és az engedélyek kezelését.

## Metódusok

### authorize()

Leírás:

-Ez a metódus határozza meg, hogy a felhasználó jogosult-e a kérés végrehajtására. Jelenleg a metódus false értéket ad vissza, ami azt jelenti, hogy a kérés nem engedélyezett.

Visszatérési érték:

-bool: false, a kérés nem engedélyezett.

Kód részlet:

```
<?php
public function authorize(): bool
{
    return false;
}
```



## rules()

Leírás:

-Ez a metódus adja vissza a kéréshez tartozó validációs szabályokat. A komment frissítéséhez a következő szabályok érvényesek:

-title: Kötelező mező, amelynek értéke szöveg típusú kell legyen.

-body: Kötelező mező, amelynek értéke szöveg típusú kell legyen.

Visszatérési érték:

-array: A validációs szabályokat tartalmazó asszociatív tömb.

Validációs szabályok:

-title: Kötelező (required), szöveg (string).

-body: Kötelező (required), szöveg (string).

Kód részlet:

```
<?php
public function rules(): array
{
    return [
        'title' => 'required|string',
        'body' => 'required|string'
    ];
}
```

## Osztály: UpdatePostsRequest

Az UpdatePostsRequest osztály a bejegyzések frissítéséhez szükséges HTTP-kérések validálásáért felelős. Ez az osztály a Laravel keretrendszer FormRequest osztályát örökli, és lehetővé teszi a bemeneti adatok validálását és az engedélyek kezelését.

## Metódusok

### authorize()

Leírás:

-Ez a metódus határozza meg, hogy a felhasználó jogosult-e a kérés végrehajtására. Jelenleg minden felhasználó számára engedélyezett (true).

Visszatérési érték:

-bool: true, ha a kérés engedélyezett.

Kód részlet:

```
<?php
public function authorize(): bool
{
    return true;
}
```

## rules()

Leírás:

-Ez a metódus adja vissza a kéréshez tartozó validációs szabályokat. A bejegyzés frissítéséhez a következő szabályok érvényesek:

-title: Kötelező mező, amelynek értéke szöveg típusú kell legyen.

-body: Kötelező mező, amelynek értéke szöveg típusú kell legyen.

Visszatérési érték:

-array: A validációs szabályokat tartalmazó asszociatív tömb.

Validációs szabályok:

-title: Kötelező (required), szöveg (string).

-body: Kötelező (required), szöveg (string).

Kód részlet:

```
<?php
public function rules(): array
{
    return [
        'title' => 'required|string',
        'body' => 'required|string'
    ];
}
```

## Osztály: UpdateRolesRequest

Az UpdateRolesRequest osztály a szerepkörök frissítéséhez szükséges HTTP-kérések validálásáért felelős. Ez az osztály a Laravel keretrendszer FormRequest osztályát örökli, és lehetővé teszi a bemeneti adatok validálását és az engedélyek kezelését.

## Metódusok

### authorize()

Leírás:

-Ez a metódus határozza meg, hogy a felhasználó jogosult-e a kérés végrehajtására. Jelenleg minden felhasználó számára engedélyezett (true).

Visszatérési érték:

-bool: true, ha a kérés engedélyezett.

Kód részlet:

```
<?php
public function authorize(): bool
{
    return true;
}
```

## rules()

Leírás:

-Ez a metódus adja vissza a kéréshez tartozó validációs szabályokat. A szerepkör frissítéséhez a következő szabályok érvényesek:

-role\_name: Kötelező mező, amelynek értéke szöveg típusú kell legyen.

Visszatérési érték:

-array: A validációs szabályokat tartalmazó asszociatív tömb.

Validációs szabályok:

-role\_name: Kötelező (required), szöveg (string).

Kód részlet:

```
<?php
public function rules(): array
{
    return [
        'role_name' => 'required|string'
    ];
}
```

# Api Végpontok dokumentáció

## Végpontok

### Felhasználók

GET /user: Visszaadja az aktuális felhasználó adatait.

GET /users: Visszaadja az összes felhasználót.

POST /users: Új felhasználó létrehozása.

PUT /users/{id}: Meglévő felhasználó frissítése.

DELETE /users/{id}: Felhasználó törlése.

## Bejegyzések

GET /posts: Visszaadja az összes bejegyzést.

POST /posts: Új bejegyzés létrehozása.

PUT /posts/{id}: Meglévő bejegyzés frissítése.

DELETE /posts/{id}: Bejegyzés törlése.

## Hozzászólások

GET /comments: Visszaadja az összes hozzászólást.

POST /comments: Új hozzászólás létrehozása.

PUT /comments/{id}: Meglévő hozzászólás frissítése.

DELETE /comments/{id}: Hozzászólás törlése.

## Kategóriák

GET /categories: Visszaadja az összes kategóriát.

POST /categories: Új kategória létrehozása.

PUT /categories/{id}: Meglévő kategória frissítése.

DELETE /categories/{id}: Kategória törlése.

## Szerepkörök

GET /roles: Visszaadja az összes szerepkört.

POST /roles: Új szerepkör létrehozása.

PUT /roles/{id}: Meglévő szerepkör frissítése.

DELETE /roles/{id}: Szerepkör törlése.

## Hitelesítés

GET /register: Regisztrációs űrlap megjelenítése.

POST /register: Felhasználó regisztrálása.

GET /login: Bejelentkezési űrlap megjelenítése.

POST /login: Felhasználó bejelentkeztetése.

POST /logout: Felhasználó kijelentkeztetése.