Fejlesztői dokumentáció





Projekt neve: PikkyPad

Készült: 2024-2025

Fejlesztői: Heszler István Bogdán Csaba

1. Bevezetés

Ez a dokumentáció a PikkyPad fejlesztői szempontú leírását tartalmazza.

Célja, hogy segítse a jelenlegi és jövőbeni fejlesztőket a rendszer megértésében, karbantartásában és továbbfejlesztésében.

1. Technológiai stack

* Frontend: Angular
* Backend: Laravel
* Adatbázis: MySQL

1. Projekt struktúra

src/

|----app/

| |---- components/

| |---- games/ az oldal játékai találhatóak itt

| |---- login/ a bejelentkezési felület található itt

| |---- pages/ különböző oldalak találhatóak itt

| |---- register/ regisztrációs felület található itt

| |---- service/ szolgáltatások találhatóak itt

1. Adatmodellek

* AuthService
  + login(data: any)

POST-kérés a /login végpontra. A data egy e-mail és jelszó pár.

* + register(data: any)

POST-kérés a /register végpontra új felhasználó regisztrálásához.

* + logout()

POST-kérés a /logout végpontra. A kérés fejlécében a Bearer token kerül elküldésre.

* + getCurrentUser()

GET-kérés a /user végpontra, amely visszaadja a bejelentkezett felhasználó adatait.

* ChildService

Ez a szolgáltatás biztosítja a gyerekekkel kapcsolatos CRUD műveletek (Create, Read, Update, Delete) kezelését, és minden kérésnél az aktuális felhasználó tokenjét használja az autentikációhoz.

* + getChildren()

lekéri az összes gyermeket.

* + addChild(data)

új gyermek hozzáadása.

* + updateChild(id, data)

meglévő gyermek adatainak módosítása.

* + deleteChild(id)

gyermekek törlése ID alapján

* GameService

A játékok kezeléséért, hozzárendeléséért, statisztikák mentéséért és egy adott gyerekhez rendelt játékok lekérdezéséért felelős szolgáltatás. Minden API-hívásnál az aktuális bejelentkezett felhasználó Bearer tokenjét használja.

**Fő metódusai:**

* + getGames()

Lekéri az összes elérhető játékot.

* + addGame(data)

Új játék létrehozása admin által.

* + updateGame(id, data)

Játék adatainak szerkesztése.

* + deleteGame(id)

Játék törlése.

* + assignGameToChild(childId, gameId)

Játék hozzárendelése adott gyermekhez.

* + removeGameFromChild(childId, gameId)

Játék leválasztása gyermekről.

* + getGamesForChild(childId)

Adott gyermekhez rendelt játékok listázása.

* + saveGameStat(childId, gameId, duration)

A játék használati statisztikájának mentése, például játékidő rögzítése.

* AdminService

Ez a szolgáltatás csak admin jogosultságú felhasználók számára érhető el. Célja a felhasználók és gyermekek kezelésének biztosítása rendszerszinten.

**Fő metódusai:**

* + getUsers() – Rendszerben regisztrált felhasználók listázása.
  + deleteUser(id) – Felhasználó törlése.
  + promoteUser(id) – Felhasználó adminná emelése.
  + getChildren() – Minden gyermek lekérése (függetlenül attól, ki a szülőjük).
  + deleteChild(id) – Bármely gyermek törlése.

1. ChildDashboardComponent

A ChildDashboardComponent az egyik legfontosabb komponens, amely felelős a gyermekek kezeléséért és a játékkal kapcsolatos interakciókért.

**Főbb feladatok:**

* **Gyermekek megjelenítése**: A gyermekek adatainak listázása, valamint új gyermek hozzáadása.
* **Játék hozzárendelése és eltávolítása**: A szülő képes hozzárendelni vagy eltávolítani játékokat a gyermekekhez.
* **Űrlapok kezelése**: A gyermekek hozzáadásához és szerkesztéséhez űrlapokat biztosít.
* **Hibakezelés és értesítések**: A Toastr segítségével az alkalmazás értesíti a felhasználót az elvégzett műveletekről.

**Kód elemzés:**

* **Űrlapkezelés**: A gyermekek hozzáadásához és szerkesztéséhez a FormBuilder segítségével dinamikusan létrehozott formák használata biztosítja a validációkat.
* **Játék hozzárendelése**: A játékok hozzárendeléséhez a gyermekekhez egy egyszerű választás szükséges, ahol a felhasználó eldöntheti, hogy melyik játékot szeretné hozzárendelni egy adott gyermekhez.
* **API hívások**: A gyermekek és játékok adatainak kezelése szolgáltatásokon keresztül történik (ChildService és GameService).

**Fontos funkciók:**

* **Gyermek hozzáadása és szerkesztése**: Az űrlap segítségével új gyermekeket lehet hozzáadni, illetve a meglévőket szerkeszteni.
* **Játékok kezelése**: A játékok a gyermekekhez rendelhetők, és eltávolíthatók, ezzel segítve a szülők számára, hogy kezeljék, melyik gyermekhez melyik játék van hozzárendelve.

1. ChildCardComponent

A ChildCardComponent komponens az egyes gyermekek megjelenítésére szolgál az alkalmazásban. Mivel a komponens csak a vizuális megjelenítéssel foglalkozik, az interakciók a szülő komponens (pl. ChildDashboardComponent) felé irányulnak.

**A komponens funkciója:**

* **Gyermekek adatainak megjelenítése**: A gyermekek egy-egy kártyán jelennek meg, amely az alapvető információkat tartalmazza, mint például név, életkor, és a hozzárendelt játékok.
* **Szerkesztés és törlés lehetősége**: A kártyán található gombok segítségével a felhasználó szerkesztheti vagy törölheti a gyermeket, vagy akár eltávolíthatja a játékot a gyermektől.
* **Játék eltávolítása**: A komponens lehetővé teszi a játékok eltávolítását a gyermekekről, ha szükséges.

**Komponens elemzés:**

**Inputok és Outputok:**

* **@Input() child: any;**: A gyermek adatait a szülő komponens küldi át a ChildCardComponent számára. Ezt a gyermek információt a kártya megjelenítésére használjuk.
* **@Output() edit = new EventEmitter<any>();**: Amikor a felhasználó szerkeszteni szeretné a gyermeket, ezt az eseményt aktiválja. Az esemény az adatokat visszaküldi a szülő komponensnek a szerkesztéshez.
* **@Output() delete = new EventEmitter<number>();**: A gyermek törléséhez az id-t átadja a szülő komponensnek, amely felelős a gyermek törléséért az adatbázisban.
* **@Output() removeGame = new EventEmitter<{ childId: number, gameId: number }>();**: Ha a felhasználó eltávolít egy játékot a gyermektől, ezt az eseményt küldi, amely tartalmazza a gyermek és a játék azonosítóját.

**Template:**

A template-ban biztosítani kell az interakciókat, például a szerkesztéshez és törléshez szükséges gombokat, valamint az egyéb információk megjelenítését.

1. AuthModalComponent

Az AuthModalComponent a bejelentkezési és regisztrációs modális ablakot kezeli. A komponens felelős a felhasználók hitelesítéséért, és az autentikációval kapcsolatos logikát kezeli, beleértve a sikeres bejelentkezés utáni navigációt. A loginForm űrlap segítségével a felhasználó be tud jelentkezni, és ha a bejelentkezés sikeres, akkor az alkalmazás a megfelelő felhasználói jogosultságok alapján navigál.

**A komponens funkciója:**

* **Bejelentkezési űrlap**: A felhasználó beírhatja a felhasználónevet és a jelszót a bejelentkezéshez.
* **Navigáció a sikeres bejelentkezés után**: A rendszer a felhasználó szerepe alapján navigál a megfelelő oldalra.
* **Hibakezelés**: A bejelentkezés vagy egyéb hiba esetén a rendszer értesíti a felhasználót.
* **Regisztrációs mód váltás**: Lehetőséget ad a regisztrációra a bejelentkezési mód mellett.

**Komponens elemzése:**

**Propertik:**

* **loginForm: FormGroup;**: Ez az űrlap csoport a bejelentkezéshez szükséges mezőkkel (felhasználónév és jelszó).
* **activeTab: 'login' | 'register' = 'login';**: Meghatározza, hogy a modális ablak bejelentkezési vagy regisztrációs nézetben van.

**Függőségek:**

* **AuthService**: Az autentikációs szolgáltatás, amely végrehajtja a bejelentkezést, a regisztrációt és más kapcsolódó funkciókat.
* **Router**: Az alkalmazás navigálásáért felelős szolgáltatás.
* **ToastrService**: A felhasználói értesítések megjelenítéséhez használt könyvtár.
* **RegisterComponent**: A regisztrációs formot tartalmazó komponens, amelyet a felhasználó a regisztrációhoz használhat.

**Módszerek:**

* **onLogin()**:
  + A bejelentkezési adatokat (felhasználónév és jelszó) elküldi az AuthService szolgáltatásnak.
  + Sikeres bejelentkezés után elmenti a token-t a localStorage-ba, majd lekéri az aktuális felhasználói adatokat.
  + A felhasználó szerepe alapján navigál a megfelelő oldalra: admin felhasználókat az admin oldalra, a többieket a privát kezdőoldalra.
  + Ha hiba történik, akkor egy értesítést küld a felhasználónak.

1. LoginComponent

A LoginComponent a felhasználó bejelentkezési folyamatát kezeli az Angular alkalmazásban. A bejelentkezéshez az email és jelszó megadását várja, és az autentikációs szolgáltatás (AuthService) segítségével ellenőrzi a bejelentkezési adatokat. Ha sikeres a bejelentkezés, a felhasználót a jogosultságának megfelelő oldalra irányítja, például az admin felületre, ha adminisztrátori szerepe van, vagy a privát kezdőlapra.

**Funkciók:**

**Form űrlap:**

* **email**: Kötelező mező, amely az email cím validálását végzi.
* **password**: Kötelező mező, amely a jelszót várja.

**Metódusok:**

* **onSubmit()**:
  + Ellenőrzi, hogy az űrlap érvényes-e.
  + Ha érvényes, a bejelentkezési adatokat elküldi az AuthService-nek.
  + A bejelentkezés után lekéri a felhasználó adatait az AuthService-től, hogy meghatározza a felhasználó szerepét.
  + Ha a felhasználó admin vagy szuperadmin, az alkalmazás az admin oldalt nyitja meg.
  + Ha más szerepe van, a privát kezdőlapra irányítja.
  + Ha bármilyen hiba történik, a rendszer értesíti a felhasználót Toastr értesítésekkel.

**Függőségek:**

* **AuthService**: Az autentikációs logikát kezeli, például a bejelentkezést és a felhasználói adatok lekérését.
* **Router**: A navigációért felelős szolgáltatás, amely a felhasználót a megfelelő oldalra irányítja.
* **ToastrService**: Felhasználói értesítések megjelenítésére szolgáló könyvtár, amely segít az információk megjelenítésében.

1. RegisterComponent

A RegisterComponent a felhasználók regisztrációját kezeli az Angular alkalmazásban. A komponens tartalmazza a szükséges mezőket, mint a felhasználói név, email, jelszó és a jelszó megerősítése. A regisztrációs űrlap helyes kitöltése után a AuthService szolgáltatás segítségével történik a regisztrációs kérés.

**Funkciók:**

**Form űrlap:**

* **name**: Kötelező mező a felhasználói névhez.
* **email**: Kötelező mező, amely az email cím validálását végzi.
* **password**: Kötelező mező, amely a jelszót várja.
* **confirm\_password**: Kötelező mező, amely a jelszó megerősítésére szolgál.

**Metódusok:**

* **onSubmit()**:
  + Ellenőrzi, hogy az űrlap érvényes-e.
  + Ha érvényes, a regisztrációs adatokat elküldi az AuthService-nek.
  + A regisztráció után értesítést küld a felhasználónak a sikeres regisztrációról (Toastr).
  + Ha hiba történik, a hibákat a válasz alapján jeleníti meg, és a felhasználónak visszajelzést ad a hiba okáról.
  + A regisztráció után az alkalmazás visszairányítja a felhasználót a kezdőlapra.

**Függőségek:**

* **AuthService**: A regisztrációt és az autentikációval kapcsolatos logikát kezeli.
* **ToastrService**: Felhasználói értesítések megjelenítésére szolgáló könyvtár.
* **Router**: A navigációt kezeli, hogy a felhasználót a megfelelő oldalra irányítsa.

1. ChildGame

A ChildGamesComponent felelős azért, hogy megjelenítse egy adott gyermekhez rendelt játékokat és lehetővé tegye a játékok indítását. Az alábbiakban részletesen bemutatom a komponens működését és a kulcsfontosságú funkcióit.

**Funkciók:**

**Alapvető felépítés:**

* **childId**: Ez a változó tárolja az aktuálisan megjelenített gyermek ID-ját, amely az URL-ből (route paraméter) kerül kinyerésre.
* **games**: Ez a tömb tárolja azokat a játékokat, amelyeket az adott gyermekhez rendeltek.

**Metódusok:**

* **ngOnInit()**:
  + A komponens betöltődésekor fut le, és kinyeri a gyermek ID-ját az URL paraméterekből.
  + Ezt követően meghívja a loadGamesForChild() metódust, hogy betöltse a gyermekhez rendelt játékokat.
* **loadGamesForChild()**:
  + A GameService segítségével lekéri az adott gyermekhez rendelt játékokat a backendből.
  + Ha a kérés sikeres, az adatokat elmenti a games tömbbe, és egy sikeres értesítést jelenít meg a felhasználónak.
  + Ha a kérés hibás, hibaüzenetet jelenít meg a felhasználónak.
* **startGame()**:
  + Ez a metódus akkor hívódik meg, amikor a felhasználó elindít egy játékot.
  + A kiválasztott játék ID-ját és a gyermek ID-ját átadja a router query paramétereiben, és navigál a /child-play oldalra, ahol a játék elindul.

1. ChildplayComponents

A ChildPlayComponent egy komplex komponens, amely kezeli a játékok lejátszását egy gyermek számára. Az alábbiakban részletesen bemutatom a komponens működését és annak kulcsfontosságú funkcióit.

**Funkciók:**

**Alapvető felépítés:**

* **games**: Tárolja a gyermekhez rendelt játékokat.
* **currentGameIndex**: Az aktuálisan lejátszott játék indexét tartalmazza.
* **timer**: A játékhoz beállított időzítő másodpercben (1 óra, vagyis 3600 másodperc).
* **intervalId**: Az időzítő kezelő ID-ja, hogy leállíthassuk, amikor a játék véget ér.

**Metódusok:**

* **ngOnInit()**:
  + Amikor a komponens betöltődik, lekéri a gyermek ID-ját a URL query paramétereiből (childId).
  + Ha a gyermek ID nem található, a felhasználó automatikusan visszairányításra kerül a /home-private oldalra.
  + A gameService.getGamesForChild() segítségével lekéri az adott gyermekhez rendelt játékokat.
  + Ha nincs játék, akkor egy információs értesítést kap a felhasználó. Ha vannak játékok, elindul a játékidőzítő.
* **currentGame()**:
  + Az aktuális játékot adja vissza az index alapján (currentGameIndex).
* **isMemoryGame()**, **isColoringGame()**, **isPuzzleGame()**:
  + Ezek a metódusok ellenőrzik, hogy az aktuális játék típusa megfelel-e a memóriajáték, színezőjáték vagy puzzle játéknak.
* **nextGame()** és **previousGame()**:
  + Ezek a metódusok segítenek navigálni a játékok között (a játékok listáján belül előre vagy hátra).
* **startTimer()**:
  + Elindítja a visszaszámláló időzítőt, ami minden másodpercben csökkenti a timer változó értékét. Ha az idő lejár, automatikusan leállítja a játékot, és a felhasználó visszakerül a home-private oldalra.
* **formatTime()**:
  + Az időt formázza percek és másodpercek formájában, például 1:30 a 90 másodperces időszakra.
* **goBack()**:
  + A felhasználó visszaléphet a kezdőoldalra, és a timer is leállítódik.

1. Home-privateComponents

A HomePrivateComponent a szülők számára biztosítja a hozzáférést a gyermekek kezeléséhez és a beállítások módosításához. Az alábbiakban részletesen ismertetem a komponens működését és annak kulcsfontosságú funkcióit.

**Funkciók és működés:**

**Alapvető felépítés:**

* **parentName**: Tárolja a szülő nevét, amit a gyermekek adatai közül olvasunk ki.
* **children**: Tárolja a szülőhöz rendelt gyermekek listáját, amelyet a ChildService segítségével töltünk le.
* **childService**: A gyermekek adatainak lekérdezésére szolgáló szolgáltatás.
* **toastr**: Az értesítések kezelésére szolgáló könyvtár, amely értesítéseket jelenít meg a felhasználónak.

**Metódusok:**

* **ngOnInit()**:
  + A komponens betöltődésekor meghívja a fetchChildren() metódust, hogy lekérje a szülőhöz tartozó gyermekeket.
* **fetchChildren()**:
  + A childService.getChildren() metódus segítségével lekéri a gyermekek adatait.
  + A válaszban található első gyermek adatából kinyeri a szülő nevét, és ezt elmenti a parentName változóba.
  + Ha a gyermekek listája üres, akkor alapértelmezettként a "Szülő" szöveget állítja be.
* **logout()**:
  + A logout() metódus a felhasználó kijelentkezését kezeli.
  + Az aktuális token-t eltávolítja a helyi tárolóból, és a felhasználót visszairányítja a kezdőoldalra.
  + Ha a kijelentkezés sikeres, akkor sikeres üzenetet jelenít meg. Ha hiba lép fel, akkor hibát jelez.
* **goToChildDashboard()**:
  + A felhasználót átirányítja a gyermekek kezelésére szolgáló oldalra (child-dashboard).

1. landingComponents

A LandingComponent a bejelentkezési és regisztrációs felületek kezelésére szolgál. A komponens célja, hogy lehetőséget adjon a felhasználóknak a bejelentkezésre és a regisztrációra, az ezekhez szükséges modálok kezelése révén. Az alábbiakban részletesen ismertetem a komponens működését.

**Funkciók és működés:**

**Alapvető felépítés:**

* **showLogin**: Logikai változó, amely meghatározza, hogy a bejelentkezési modál legyen látható vagy sem.
* **showRegister**: Logikai változó, amely meghatározza, hogy a regisztrációs modál legyen látható vagy sem.
* **openLogin()**: Ez a metódus a bejelentkezési modál megjelenítésére szolgál. Ha meghívják, a bejelentkezési modál láthatóvá válik, és a regisztrációs modál eltűnik.
* **openRegister()**: Ez a metódus a regisztrációs modál megjelenítésére szolgál. Ha meghívják, a regisztrációs modál láthatóvá válik, és a bejelentkezési modál eltűnik.
* **closeModals()**: Ez a metódus eltünteti mindkét modált, beállítva a showLogin és showRegister változókat false értékre.

1. Admin-childrenComponents

A AdminChildrenComponent az adminisztrátori felület egyik komponense, amely a gyermekek kezelésére szolgál. Itt a rendszergazdák képesek listázni a gyermekeket, valamint törölni őket. A komponens az adminisztrációs felület kezelésére szolgál, és az alábbiakban részletesen ismertetem a működését.

**Funkciók és működés:**

**Alapvető felépítés:**

* **children**: Ez a változó tárolja az összes gyermek adatait, amelyeket az adminisztrátor a felületen megtekinthet.
* **loadChildren()**: Ez a metódus felelős a gyermekek adatainak betöltéséért. Az adminService.getChildren() segítségével lekérdezi az adatokat a backendről, és a választ a children tömbbe helyezi.
* **deleteChild(id: number)**: Ez a metódus a gyermek törlésére szolgál. Ha a felhasználó megerősíti a törlést egy visszaigazoló párbeszédablakban, akkor az adminService.deleteChild(id) metódus segítségével törli a gyermeket az adatbázisból, és újratölti a gyermekek listáját.

**Toastr értesítések:**

* **Sikeres gyermek törlés:** Ha a gyermek sikeresen törlődik, akkor egy értesítést jelenít meg a felhasználónak (this.toastr.success()).
* **Hiba törléskor:** Ha a törlés során hiba történik, akkor hibát jelez (this.toastr.error()).
* **Gyermekek betöltése hiba:** Ha a gyermekek adatainak betöltése nem sikerül, akkor egy hibaüzenetet jelenít meg.

1. Admin-gameComponents:

A AdminGameComponent egy adminisztrátori felület, amely lehetővé teszi a játékok kezelését, beleértve azok hozzáadását, frissítését és törlését. Az alábbiakban bemutatom a komponens működését és logikáját.

**Funkciók és működés:**

**Formuláris kezelés:**

A komponens egy **gameForm** nevű űrlapot használ, amely három mezőt tartalmaz:

* **title**: A játék neve (kötelező mező)
* **description**: A játék leírása (nem kötelező)
* **category**: A játék kategóriája (kötelező mező)

Az űrlapot a **ReactiveFormsModule** segítségével hozjuk létre, és validáljuk a kötelező mezőket.

**Alapvető működés:**

* **loadGames()**: Betölti a játékokat a backendből az gameService.getGames() metódus hívásával, és a választ a **games** tömbben tárolja.
* **onSubmit()**: Az űrlap adatainak mentésére szolgál. Ha az űrlap érvényes, akkor:
  + Ha van **editId**, akkor egy meglévő játékot frissítünk az updateGame() metódussal.
  + Ha nincs **editId**, akkor új játékot adunk hozzá az addGame() metódussal.
  + Minden sikeres művelet után a játékok újratöltődnek, és az űrlap törlődik.
* **editGame(game: any)**: Ez a metódus egy meglévő játék szerkesztéséhez szolgál. Ha a játékot szerkesztjük, akkor az űrlap mezőit az adott játékkal tölti fel.
* **deleteGame(id: number)**: A játék törlésére szolgáló metódus. A törlés előtt megerősítő párbeszédablak jelenik meg, majd az deleteGame() metódus törli a játékot a backendből.
  +  **Sikeres mentés vagy frissítés:** A felhasználó értesítést kap, hogy a játék sikeresen hozzá lett adva vagy frissítve (this.toastr.success()).
  +  **Hiba a mentés vagy frissítés során:** Ha a mentés vagy frissítés nem sikerül, hibát jelez (this.toastr.error()).
  +  **Sikeres törlés:** A törlés sikeres, és az admin felhasználó értesítést kap.
  +  **Hiba törléskor:** A törlés során fellépett hiba esetén hibás értesítést kap.

1. Admin-layoutComponents

A AdminLayoutComponent egy adminisztrátori felületet biztosít, amely tartalmazza a napra kész időt és dátumot, valamint a bejelentkezett felhasználó nevét, és lehetőséget ad a kijelentkezésre. Itt van, hogyan működik a komponens:

**Funkciók és működés:**

* **Idő és Dátum Frissítés:**
  + A komponens folyamatosan frissíti az időt és a dátumot a setInterval() függvény használatával, így mindig naprakész információt mutat.
  + Az idő másodpercenként frissül a currentTime változóban, és a dátum is az currentDate változóban.
* **Bejelentkezett Felhasználó Neve:**
  + A komponens a localStorage-ban tárolt tokent ellenőrzi, és ha a token jelen van, akkor egy HTTP kérésen keresztül lekéri az aktuális felhasználó nevét az AuthService.getCurrentUser() segítségével.
  + Ha a kérés sikeres, beállítja a felhasználó nevét a userName változóba, különben az alapértelmezett "Admin" nevet használja.
* **Kijelentkezés:**
  + A logout() metódus felelős a felhasználó kijelentkeztetéséért. A kijelentkezéshez a következő lépések történnek:
    - A token törlésre kerül a localStorage-ból.
    - A felhasználó átirányításra kerül a kezdőoldalra (/).
    - Ha a logout kérés hiba történik, az alkalmazás akkor is a kezdőoldalra navigál.

1. Admin-userComponents:

A AdminUserComponent a felhasználók kezelésére szolgáló komponens, amely lehetővé teszi a felhasználók betöltését, törlését és adminisztrátori jogosultságokkal történő frissítését. Itt van, hogy hogyan működik a komponens:

**Funkciók és működés:**

* **Felhasználók Betöltése (loadUsers()):**
  + Az ngOnInit életciklus-horogban meghívódik a loadUsers() metódus, amely az AdminService.getUsers() szolgáltatáson keresztül lekéri a felhasználói adatokat.
  + A válasz (res) sikerességétől függően a felhasználók listáját frissíti. Ha hiba történik, egy hibaüzenetet jelenít meg a ToastrService segítségével.
* **Felhasználó Törlése (deleteUser(id: number)):**
  + A törlés előtt megerősítést kér a felhasználótól (confirm), hogy biztosan törölni akarja-e a felhasználót.
  + A törlés sikeressége után a felhasználó listája újratöltődik. Ha hiba történik, hibaüzenetet jelenít meg.
* **Felhasználó Adminisztrátorrá Lépése (promoteUser(id: number)):**
  + Hasonlóan a törléshez, a promoteUser() metódus először megerősítést kér a felhasználótól, hogy adminná akarja-e léptetni a felhasználót.
  + A frissítés sikeressége után a felhasználók listája újratöltődik, és egy visszajelző üzenet jelenik meg.

1. ColoringGameComponents

A ColoringGameComponent egy egyszerű színezős játék komponens, amely lehetővé teszi a felhasználók számára, hogy különböző színekkel színezzék ki a képeken található területeket.

**Funkciók és működés:**

* **Színek Lista (colors):**
  + A komponens colors változója tartalmazza a lehetséges színeket, amelyeket a felhasználó kiválaszthat.
  + Az egyes színek hexadecimális kódok formájában vannak megadva.
* **Kiválasztott Szín (selectedColor):**
  + Az aktuálisan kiválasztott szín tárolására szolgáló változó, amely alapértelmezetten a piros (#FF0000).
* **Színezett Területek (selectedColors):**
  + A selectedColors objektum tárolja, hogy mely területek (azonosítóval rendelkező elemek) milyen színnel vannak színezve.
  + Az objektum kulcsa a pathId (ami a terület azonosítója), és az értéke a kiválasztott szín.
* **Szín Kiválasztása (onColorClick(pathId: string)):**
  + Ez a metódus frissíti a selectedColors objektumot az adott pathId azonosítóhoz tartozó színnel.
  + Ez akkor történik, amikor a felhasználó egy területre kattint, így az a terület az aktuálisan kiválasztott színt kapja.

1. PuzzleGameComponents

A PuzzleGameComponent egy egyszerű, interaktív puzzle játékot valósít meg Angularban. A játékban a felhasználó a képkockákat (puzzle darabokat) húzhatja a megfelelő helyekre, és ellenőrizhetjük, hogy helyesen rendezték-e őket.

**Funkciók és működés:**

* **Képek és Puzzle Darabok:**
  + A imageUrls változóban tároljuk az elérhető képeket, amelyekből véletlenszerűen választunk egyet, amikor a játék elindul.
  + A puzzlePieces tömb tartalmazza a puzzle darabok pozícióit, valamint azok megfelelő helyét a rácsban.
* **Rács Beállításai:**
  + A puzzle rácsának méretét a gridSize változó határozza meg (például 3x3, tehát 9 darab puzzle rész lesz).
  + A pieceSize változó a puzzle darabok méretét állítja be (100px).
* **Kép Kezelése:**
  + Az imageUrl változó a kiválasztott kép URL-jét tárolja.
  + A getRandomImage metódus egy véletlenszerű képet választ a képek közül.
* **Puzzle Darabok Inicializálása:**
  + Az initializePuzzle metódus létrehozza a puzzle darabokat, meghatározza azok pozícióit, és a darabokat összekeveri a shufflePieces metódussal, hogy a játék kihívást jelentsen.
* **Húzás és Áthelyezés:**
  + A startDrag metódus elindítja a darab húzását, rögzíti a kezdő pozíciót, és eseménykezelőt ad hozzá a mozgás és az elengedés figyelésére.
  + A dragPiece metódus frissíti a darab pozícióját a húzás közben.
  + A stopDrag metódus eltávolítja az eseménykezelőket, amikor a felhasználó elengedi a darabot.
* **Megoldás Ellenőrzése:**
  + A checkSolution metódus végignézi, hogy a puzzle darabok megfelelő helyeken vannak-e. Ha minden darab a helyére került, akkor egy gratuláló üzenetet jelenít meg.

1. MemorygameComponents

A MemoryGameComponent egy memóriajátékot valósít meg, ahol a felhasználó párba kell rendezi a kártyákat úgy, hogy a megfelelő párok fel legyenek fordítva. A játék célja, hogy minden párt megtaláljunk, amíg minden kártya a helyére nem kerül.

**Funkciók és működés:**

* **Kártyák és játékállapot:**
  + A cards tömb tárolja a kártyák adatait, beleértve a kép URL-jét (url), hogy a kártya párosítva van-e (matched), és hogy a kártya le van-e fordítva (flipped).
  + A flippedCards tömb tartalmazza azokat a kártyákat, amelyeket a felhasználó éppen megfordított.
* **Játék Állapotok:**
  + A allMatched változó ellenőrzi, hogy minden kártyapár helyesen megtalálásra került-e (nyert a játékos).
* **Játék Inicializálása:**
  + Az initializeCards metódus inicializálja a kártyákat. Az elérhető képek listáját (imageUrls) duplázzuk, hogy minden képből két kártya legyen. A kártyák véletlenszerűen vannak elrendezve a játéktáblán a sort(() => Math.random() - 0.5) módszerrel.
* **Kártyák Megfordítása:**
  + A flipCard metódus felelős a kártyák megfordításáért. Amikor egy kártyát megfordítunk, a flipped érték true-ra változik, és a kártya adatai bekerülnek a flippedCards tömbbe.
  + Ha két kártya van megfordítva, a checkForMatch metódus ellenőrzi, hogy a két kártya azonos képet tartalmaz-e.
* **Párok Ellenőrzése:**
  + Ha a két kártya egyezik, a matched értéke true-ra változik, és ellenőrizzük, hogy minden kártya párosítva lett-e a checkForWin metódussal.
  + Ha a két kártya nem egyezik, akkor egy kis idővel visszafordítjuk őket (1 másodperc), és az flippedCards tömb kiürül.
* **Játék Újraindítása:**
  + A restartGame metódus újraindítja a játékot, alaphelyzetbe állítja a kártyákat, a flipped kártyákat és a nyert állapotot, majd újra inicializálja a kártyákat.

Backend

1. AdminController

* **index()**
  + **Leírás:** Ez a metódus az összes felhasználót és azok gyermekeit lekérdezi. A User::with('children')->get() kód a felhasználókat és az ahhoz tartozó gyerekeket egyaránt betölti.
  + **Működés:**
    - Lekérdezi a User modelleket, és minden egyes felhasználóhoz betölti a kapcsolódó gyerekeket a children kapcsolaton keresztül.
    - Válaszként JSON formátumban visszaadja a sikeres adatokat és egy üzenetet, ami megerősíti a lekérdezés sikerességét.
  + **destroy($id)**
    - **Leírás:** Egy adott felhasználó törlésére szolgál az admin számára. A $id paraméter alapján keresi meg a felhasználót a User::find($id) segítségével.
    - **Működés:**
    - A metódus megkeresi a felhasználót az adatbázisban az adott $id alapján.
    - Ha nem található a felhasználó, akkor 404-es hibával tér vissza, és egy hibaüzenetet küld.
    - Ha megtalálja a felhasználót, akkor törli azt az adatbázisból, és egy sikeres választ ad vissza.
  + **promoteToAdmin($id)**
    - **Leírás:** A metódus célja, hogy egy adott felhasználót adminisztrátorrá léptessen. Ha a felhasználó már admin, nem történik változás.
    - **Működés:**
      * Megkeresi a felhasználót az $id alapján.
      * Ha a felhasználó nem található, 404-es hibát küld.
      * Ha a felhasználó már admin, egy hibaüzenetet küld vissza.
      * Ha a felhasználó nem admin, akkor módosítja a szerepkört (role = 'admin'), és elmenti az adatbázisba.
  + **listChildren()**
    - **Leírás:** A metódus lekérdezi az összes gyermeket, és az ahhoz tartozó szülő adatokat.
    - **Működés:**
      * A Child::with('user')->get() kód segítségével az összes gyermeket és azok szüleit lekérdezi.
      * Válaszként JSON formátumban visszaadja a sikeres adatokat és egy üzenetet.
  + **deleteChild($id)**
* **Leírás:** Egy adott gyermek törlését végzi el az admin számára a megadott $id alapján.
* **Működés:**
  + A metódus megkeresi a gyermeket az adatbázisban az adott $id alapján.
  + Ha nem található a gyermek, 404-es hibát küld vissza.
  + Ha megtalálja a gyermeket, akkor törli azt az adatbázisból, és egy sikeres választ ad vissza.

1. AuthController

* **register(RegisterRequest $request)**
  + **Leírás:** Ez a metódus a felhasználók regisztrációját kezeli. A regisztrációs adatokat a RegisterRequest validálja, majd egy új felhasználót hoz létre az adatbázisban.
  + **Működés:**
    - A metódus először validálja a regisztrációs adatokat a RegisterRequest osztály segítségével.
    - Ezután létrehoz egy új felhasználót az User::create() metódus használatával, és az adatokat a name, email, és password mezőkkel menti. A jelszó titkosításához a bcrypt() függvényt használja.
    - Ha a regisztráció sikeres, visszaadja a felhasználó adatait és egy üzenetet.
* **login(LoginRequest $request)**
  + - **Leírás:** A metódus a felhasználók bejelentkezését kezeli. Az Auth::attempt() metódust használja az autentikációhoz, és ha sikeres, akkor egy hozzáférési token-t generál.
  + **Működés:**
    - A metódus először validálja a bejelentkezési adatokat a LoginRequest osztály segítségével.
    - Az Auth::attempt() metódus próbálja autentikálni a felhasználót az email és jelszó alapján.
    - Ha a bejelentkezés sikeres, a rendszer generál egy token-t a felhasználó számára a createToken() metódus segítségével.
    - A válaszban visszaküldi a felhasználó nevét és a generált token-t.
* **logout()**
  + **Leírás:** A metódus a felhasználó kijelentkezését kezeli, és törli az éppen aktív hozzáférési token-t.
  + **Működés:**
    - Az auth('sanctum')->user() hívás segítségével lekéri az aktuálisan bejelentkezett felhasználót.
    - A currentAccessToken()->delete() metódussal törli a felhasználó jelenlegi hozzáférési token-jét.
    - A válaszban visszaküldi a felhasználó nevét és egy sikeres kijelentkezési üzenetet.
* **getUsers()**
  + **Leírás:** A metódus az összes felhasználót lekérdezi az adatbázisból és visszaadja azokat.
  + **Működés:**
    - Az User::all() metódussal lekéri az összes felhasználót az adatbázisból.
    - A lekérdezett adatokat visszaadja JSON formátumban.

1. **BannerController**
   * + **getLoginCounter($name)**

* **Cél:** Visszaadja az adott felhasználó login\_counter értékét (hányszor próbált bejelentkezni).
* **Működés:**
  1. Lekéri az adott nevű felhasználót.
  2. Visszaküldi a bejelentkezési számlálót.
* **Használat:** Frontend lekérdezheti, hogy hány próbálkozása volt már a felhasználónak.
  + - **setLoginCounter($name)**
* **Cél:** Megnöveli az adott felhasználó login\_counter értékét eggyel.
* **Működés:**
  1. Lekéri a felhasználót név alapján.
  2. Az increment() metódussal növeli a számlálót.
* **Használat:** Sikertelen bejelentkezés után meghívható, hogy naplózza a próbálkozást.
  + - **resetLoginCounter($name)**
* **Cél:** Visszaállítja a login\_counter értékét 0-ra.
* **Működés:**
  1. Lekéri a felhasználót.
  2. Nullázza a számlálót, majd menti.
* **Használat:** Sikeres bejelentkezés után lehet resetelni.
  + - **getBanningTime($name)**
* **Cél:** Lekérdezi, mikor jár le a felhasználó ideiglenes tiltása (banning\_time).
* **Működés:**
  1. Lekéri a felhasználót.
  2. Visszaadja a banning\_time mezőt.
* **Használat:** A frontend ezzel eldöntheti, hogy a felhasználó még tiltva van-e.
  + - **setBanningTime($name)**
* **Cél:** Kitiltási idő beállítása az aktuális idő +1 perc-re.
* **Működés:**
  1. A Carbon::now()->addMinutes(1) segítségével beállítja a jövőbeli tiltási időt.
  2. Mentés után a rendszer már nem engedi be a felhasználót 1 percig (frontend vagy middleware tudja ezt figyelni).
* **Használat:** Pl. 3 hibás próbálkozás után a rendszer kitilthatja a felhasználót.
  + - **resetBanningTime($name)**
* **Cél:** A kitiltás idejének törlése (null értékre állítása).
* **Működés:**
  1. A banning\_time mezőt nullázza.
  2. Ezáltal a felhasználó ismét be tud jelentkezni.
* **Használat:** Lejárat után vagy admin által manuálisan.

**Technikai háttér / Előfeltételek**

A users táblában kell legyen két mező:

login\_counter (int) – bejelentkezési próbálkozások száma.

banning\_time (datetime | nullable) – a blokkolás lejárati ideje.

24**. ChildController**

* **ChildController – Részletes leírás**
* **Jogosultságkezelés**
* **Szülő**: csak a saját gyermekeit láthatja és kezelheti.
* **Admin**: minden gyermekhez és felhasználóhoz hozzáfér.
  + - **child\_query(Request $request)**
* **Cél**: A felhasználó gyermekei (vagy admin esetén minden gyermek) lekérdezése.
* **Viselkedés**:
  + Ha admin: minden gyermek + szülő neve (with('user:id,name'))
  + Ha nem admin: csak a saját gyermekei + játékaik (with('games'))
* **Válasz**: JSON a gyermekek adataival.
* **add\_child(Request $request)**
  + **Cél**: Új gyermek létrehozása a bejelentkezett szülő számára.
  + **Validáció**:
    - Név: kötelező, max 255 karakter
    - Életkor: 3 és 9 év között kötelező
  + **Mentés**: Az auth()->user()->id alapján beállítja a szülőt.
  + **Válasz**: A létrehozott gyermek adatai.
* **update(Request $request, $id)**
  + **Cél**: Egy adott gyermek adatainak frissítése.
  + **Jogosultság**:
    - Csak a saját gyermeket szerkesztheti a szülő, admin bárkit.
  + **Validáció**: név és életkor opcionálisan frissíthető.
  + **Válasz**: Frissített gyermek adat.
* **destroy(Request $request, $id)**
* **Cél**: Gyermek törlése.
* **Jogosultság**: Csak saját gyermeket, kivéve ha admin.
* **Válasz**: Sikeres törlés üzenet.
  + - **assignGame(Request $request, $childId)**
* **Cél**: Egy játék hozzárendelése egy gyermekhez (kapcsolat a pivot táblán keresztül).
* **Validáció**: game\_id léteznie kell a games táblában.
* **Művelet**: games() kapcsolaton keresztül .attach()
* **Válasz**: Sikeres hozzárendelés.
* **6. getGamesForChild($id)**
  + **Cél**: Egy adott gyermekhez tartozó játékok lekérdezése.
  + **Művelet**: Child::with('games')
  + **Válasz**: A gyermekhez tartozó játékok listája.
* **7. removeGame($childId, $gameId)**
  + **Cél**: Egy játék eltávolítása a gyermekhez rendelt játékok közül.
  + **Jogosultság**: Csak a saját gyermekekhez tartozó játék távolítható el.
  + **Művelet**: games()->detach($gameId)
  + **Válasz**: Sikeres eltávolítás.

25.GameController

**Jogosultság**

Jelen implementáció nem tartalmaz explicit jogosultságellenőrzést (middleware vagy role check), ezért minden bejelentkezett felhasználó hozzáférhet a végpontokhoz. (Ez később bővíthető.)

* **game\_query()**
  + **Leírás**: Az összes játék lekérdezése az adatbázisból.
* **add\_game(Request $request)**
  + **Leírás**: Új játék létrehozása.
  + **Validáció**:
  + title: kötelező, string, max 255 karakter
  + description: opcionális, string
  + category: kötelező, string
  + **Művelet**: A validált adatok alapján új rekord jön létre a games táblában.
* **update(Request $request, $id)**
  + **Leírás**: Meglévő játék adatainak frissítése.
  + **Validáció**:
  + title: opcionális, string, max 255 karakter
  + description: opcionális, string
  + age\_group: opcionális, integer
  + **Feltétel**: Ha a játék nem található, 404-es válasz.
  + **Művelet**: Az only(...) alapján megadott mezők frissítése a rekordban.
* **destroy($id)**
  + **Leírás**: Egy játék törlése az adatbázisból.
  + **Feltétel**: Ha a játék nem található, 404-es válasz.
  + **Művelet**: .delete() meghívása az adott játékon.

26.ResponseController

* **sendResponse($data, $message)**
  + **Cél**: Sikeres válasz visszaküldése egységes JSON struktúrában.
* **sendError($error, $errorMessages = [], $code = 404)**
  + **Cél**: Hibaüzenet visszaküldése hibás kérés esetén.
  + **Paraméterek:**
    - $error: fő hibaüzenet (pl. "Nem található", "Érvénytelen adatok")
    - $errorMessages: opcionális részletes hiba (tömb vagy string)
    - $code: HTTP státuszkód (default: 404)

27.SuperAdminController

* **listAdmins()**
  + **Cél**: Összes admin és superadmin felhasználó lekérdezése.
* **promoteToSuperAdmin($id)**
  + **Cél**: Egy meglévő admin szerepű felhasználó előléptetése superadmin-ná.
  + **Viselkedés:**
    - Ellenőrzi, hogy létezik-e a felhasználó
    - Megnézi, hogy tényleg admin-e (csak ő léptethető elő)
    - Módosítja a role mezőt superadmin-ra

AdminMiddleware

**Cél:**

* Korlátozni a hozzáférést csak azokra a felhasználókra,
  + - * + akik admin jogosultsággal rendelkeznek.
*  Ellenőrzi, hogy van-e bejelentkezett felhasználó ($user)
*  Meghívja az $user->isAdmin() metódust (jó gyakorlat az ilyen szerepellenőrzést modellekbe tenni!)

Models

* Child
* **Child modell – Funkcionális áttekintés**
* A Child modell a **gyermekek adatait** reprezentálja az adatbázisban. Laravelben ez egy **Eloquent ORM modell**, ami azt jelenti, hogy automatikusan összeköti a kódban használt osztályokat az adatbázis táblákkal.
* **Kapcsolatok**
  + **user()**
    - **Ez a kapcsolat azt mondja:  
      Minden gyermek egy szülőhöz (felhasználóhoz) tartozik.**
    - Funkció:
      * Lehetővé teszi, hogy így lekérd a szülőt egy gyereknél:
  + **games()**
    - Ez egy **több-a-többhöz (many-to-many)** kapcsolat.  
      Egy gyermeknek lehet több játéka, és egy játékot több gyerek is játszhat.
    - Laravel feltételezi, hogy ehhez egy **pivot tábla** van, itt: child\_game.
    - Funkció:
      * Lekérdezés
      * Hozzárendelés
      * Eltáolítás
* Game
  + A Game modell az alkalmazásban a **játékokat** képviseli, és az adatbázis games táblájával van összekötve. Laravel Eloquent ORM segítségével egyszerűen lehet vele dolgozni.

| **Tulajdonság / metódus** | **Funkció** |
| --- | --- |
| fillable | Meghatározza, milyen mezőket lehet biztonságosan menteni (pl. title, description, category). |
| children() | Több gyerekhez is hozzárendelhető egy játék (belongsToMany) kapcsolaton keresztül. |

* Statistics
* Ez a modell a **gyermekek játékstatisztikáit** tárolja, tehát egy-egy játékhoz kapcsolódó eredményeket (pl. pontszám, játékidő).
* **Összekötő modellként is működik**
  + Kapcsolat van:
  + egy gyermekkel (child\_id)
  + egy játékkal (game\_id)
* **Mire való?**

| **Mező** | **Jelentés** |
| --- | --- |
| child\_id | Melyik gyermekhez tartozik ez a statisztika |
| game\_id | Melyik játékhoz tartozik az adat |
| score | Elért pontszám |
| play\_time | Mennyi ideig játszott (pl. másodpercben vagy percben) |

* User

.

* **Autentikáció**
  + Ez az osztály az Authenticatable osztályból öröklődik, tehát:
  + támogatja a bejelentkezést (jelszóval),
  + használható Laravel Sanctum API tokenes autentikációval

**Lehetséges szerepkörök**

* parent: normál felhasználó (szülő), csak a saját gyerekeit látja, kezeli
* admin: több jogosultsággal rendelkezik, pl. minden gyermeket lát
* superadmin: adminokat is kezelhet, másokat adminná tehet stb.

| **Funkció** | **Jelentés** |
| --- | --- |
| children() | A felhasználóhoz tartozó gyerekek |
| isAdmin() | Ellenőrzi, hogy admin-e vagy superadmin |
| isSuperAdmin() | Csak akkor igaz, ha superadmin |
| fillable, hidden | API válaszok és tömeges mentés biztonsága |
| casts() | Automatikus jelszó hashelés és dátumkezelés |

CONFIG

* App

**'name' => env('APP\_NAME', 'Laravel')**

* Az alkalmazás neve.
* Ezt látod pl. e-mailekben, vagy ha kiíratod valahol az app nevét.
* .env fájlban: APP\_NAME=MyCoolApp

**'env' => env('APP\_ENV', 'production')**

* Az alkalmazás futási környezete: local, staging, production, stb.
* Hasznos, hogy bizonyos viselkedést (pl. debug, cache) másként kezeljen különböző környezetekben.

**'debug' => env('APP\_DEBUG', false)**

* Ha true, akkor részletes hibaüzeneteket kapsz (stack trace), fejlesztéshez nagyon hasznos.
* Ha false, akkor csak általános hibák jelennek meg (pl. 500 - „Valami elromlott”), éles környezetben ajánlott.

**'url' => env('APP\_URL', 'http://localhost')**

* Az alkalmazás fő URL-je, pl. http://localhost vagy https://example.com
* Laravel ezt használja automatikusan linkek, assetek generálásához Artisan parancsokban vagy mail linkekben.

**'timezone' => env('APP\_TIMEZONE', 'UTC')**

* Az alapértelmezett időzóna az alkalmazásban (PHP date() és Carbon osztály használatához).

**'locale' => env('APP\_LOCALE', 'en')**

* Az alkalmazás alapértelmezett nyelve, pl. hu magyarra.

**'fallback\_locale'**

* Ha egy fordítás hiányzik az APP\_LOCALE szerinti nyelven, akkor ezt használja helyettesítésként.

**'faker\_locale'**

* Faker (adatgenerátor) mely nyelvet használjon (pl. magyar nevekhez hu\_HU).

**'key' => env('APP\_KEY')**

* Titkosítási kulcs. A Laravel ezzel titkosít jelszavakat, tokeneket stb.
* 32 karakter hosszú kulcs, amely automatikusan generálódik a php artisan key:generate paranccsal.

**'maintenance' blokk**

* Laravel 10-ben a karbantartási mód már nemcsak fájlalapú, hanem **több gépen is kezelhető** (pl. cache driveren keresztül).
* 'driver' => 'file' vagy 'cache'
* 'store' => 'database' esetén például adatbázis tároló lesz a forrás.
* Auth

| **Rész** | **Funkció** |
| --- | --- |
| defaults | Alapértelmezett guard és jelszó reset beállítás |
| guards | Autentikációs mechanizmusok (pl. session vagy token alapú) |
| providers | Milyen módon, milyen modellből jön a felhasználó |
| passwords | Jelszó reset működésének részletei |
| password\_timeout | Mennyi ideig él egy jelszómegerősítés |

* Cache

| **Rész** | **Funkció** |
| --- | --- |
| default | Alapértelmezett cache tároló beállítása |
| stores | Különböző gyorsítótár tárolók és azok konfigurálása |
| prefix | Cache kulcsok előtagjának beállítása |

* Database

| **Rész** | **Funkció** |
| --- | --- |
| default | Az alapértelmezett adatbázis kapcsolat |
| connections | Különböző adatbázis kapcsolat beállítások |
| migrations | A migrációk tárolására használt tábla konfigurációja |
| redis | Redis beállítások cache és adatbázis használatra |

* Filesystems

| **Rész** | **Funkció** |
| --- | --- |
| default | Az alapértelmezett fájlrendszer beállítása |
| disks | Különböző fájlrendszerek konfigurációja (pl. local, s3) |
| links | A szimbolikus linkek beállítása a fájlok nyilvános kiszolgálására |

* Logging

| * **Rész** | * **Funkció** |
| --- | --- |
| * default | * Az alapértelmezett naplócsatorna |
| * deprecations | * A deprekált funkciók naplózása és figyelmeztetések |
| * channels | * Különböző naplócsatornák beállításai (pl. single, daily, slack) |

* Mail

| **Rész** | **Funkció** |
| --- | --- |
| default | Az alapértelmezett e-mail szállító, amelyet az alkalmazás használ. |
| mailers | A különböző e-mail szállítók konfigurációja (SMTP, SES, Postmark, stb.) |
| from | A globális "From" cím és név beállítása minden e-mailhez. |

* Queue

| **Rész** | **Funkció** |
| --- | --- |
| default | Az alapértelmezett queue kapcsolat beállítása. |
| connections | Különböző queue backendek (sync, database, beanstalkd, stb.) beállításai. |
| batching | Job batching lehetőségek konfigurálása. |
| failed | A sikertelen queue feladatok kezelése és tárolása. |

* Sanctum

| **Rész** | **Funkció** |
| --- | --- |
| stateful | A domainek listája, amelyek állapotot tartanak fenn az API kéréseknél. |
| guard | Az autentikációs őr beállítása. Alapértelmezett: web. |
| expiration | A tokenek lejárati idejének beállítása. Alapértelmezett: null (nem járnak le). |
| token\_prefix | A tokenek előtagjának beállítása, hogy biztonságosabbá tegye őket. |
| middleware | A szükséges köztes szoftverek (middleware) konfigurálása. |

* Service

| **Szolgáltatás** | **Beállítások** | **Környezeti változók (env)** |
| --- | --- | --- |
| **Postmark** | Email küldés API kulcsa | POSTMARK\_TOKEN |
| **SES** | AWS email küldő szolgáltatás | AWS\_ACCESS\_KEY\_ID, AWS\_SECRET\_ACCESS\_KEY, AWS\_DEFAULT\_REGION |
| **Resend** | Email küldő API kulcsa | RESEND\_KEY |
| **Slack** | Slack értesítések API kulcsa és csatorna | SLACK\_BOT\_USER\_OAUTH\_TOKEN, SLACK\_BOT\_USER\_DEFAULT\_CHANNEL |

* Session

| **Beállítás** | **Leírás** |
| --- | --- |
| **driver** | A session adat tárolás módja (például fájlok, adatbázis, Redis) |
| **lifetime** | A session élettartama percben |
| **expire\_on\_close** | A session lejárat, ha a böngészőt bezárják |
| **encrypt** | Session adatok titkosítása |
| **files** | Ha fájlokban tárolod a session adatokat, itt található a fájlok helye |
| **connection** | Az adatbázis kapcsolat a session tárolásához (ha adatbázis driver van beállítva) |
| **table** | Az adatbázis tábla neve, ahol a session adatok tárolódnak (ha adatbázis driver van beállítva) |
| **cookie** | A session cookie neve |
| **secure** | Biztonságos cookie-k beállítása (HTTPS kapcsolat) |
| **http\_only** | A cookie hozzáférés JavaScript-ből való tiltása |
| **same\_site** | Same-Site cookie beállítások (CSRF támadások ellen) |

ADATBÁZIS

* Migration
  + User-table

Ez a migráció három adatbázistáblát hoz létre:

| **Tábla** | **Funkciója** |
| --- | --- |
| users | A regisztrált felhasználók adatait tárolja |
| password\_reset\_tokens | Jelszó-visszaállítási tokeneket tárol |
| sessions | Aktív felhasználói munkameneteket (session-öket) tárol |

* + Cache-table

Ez a migráció **két új táblát** hoz létre az adatbázisban:

| **Tábla** | **Funkció** |
| --- | --- |
| cache | Általános alkalmazás cache adatainak tárolása |
| cache\_locks | Párhuzamos cache módosítások zárolásának kezelése (lockok) |

* + **jobs-table**

**Mit csinál ez a migráció?**

| **Tábla neve** | **Funkció** |
| --- | --- |
| jobs | Folyamatban lévő vagy várakozó háttérfeladatok tárolása |
| job\_batches | Több feladat együttes futásának (batch) kezelése |
| failed\_jobs | Elhibázott háttérfeladatok naplózása hibakezeléshez |

* + Children-table

| **Mező** | **Típus** | **Leírás** |
| --- | --- | --- |
| id | BIGINT (auto increment) | Gyermek egyedi azonosítója. |
| user\_id | FOREIGN KEY (users tábla) | Kapcsolat a szülő (felhasználó) adataihoz. |
| name | VARCHAR(255) | Gyermek neve. |
| age | INTEGER | Gyermek életkora. |
| timestamps | created\_at, updated\_at automatikusan hozzáadva. |  |

* + GAMES-TABLE

| **Mező** | **Típus** | **Leírás** |
| --- | --- | --- |
| id | BIGINT (auto increment) | Játék egyedi azonosítója. |
| title | VARCHAR(255) | A játék címe, például: *"Memória játék"*. |
| description | TEXT | A játék rövid leírása, mi történik a játékban. |
| category | VARCHAR(255) | A játék kategóriája, például: *"logikai"*, *"színezős"*. |
| timestamps | created\_at, updated\_at automatikusan hozzáadva. |  |

* + Personal-acces-tokens

**Mi kerül létrehozásra?**

| **Mező** | **Típus** | **Leírás** |
| --- | --- | --- |
| id | BIGINT | Egyedi azonosító (auto-increment). |
| tokenable\_id | BIGINT | A token tulajdonosának az ID-ja (pl. user ID). |
| tokenable\_type | STRING | A modell típusa (pl. App\Models\User). |
| name | STRING | Token megnevezése (pl. "Web böngésző", "Mobil app"). |
| token | STRING(64) | Maga a token értéke (hash formában, egyedi). |
| abilities | TEXT | Milyen jogokat ad a token (pl. ["read", "write"]). |
| last\_used\_at | TIMESTAMP | Mikor használták utoljára a tokent? |
| expires\_at | TIMESTAMP | Mikor jár le a token? |
| created\_at, updated\_at | TIMESTAMP | Laravel automatikus időbélyegei. |

* + Child-game-table

| **Mező** | **Típus** | **Leírás** |
| --- | --- | --- |
| id | BIGINT | Egyedi azonosító. |
| child\_id | BIGINT | Hivatkozás a gyerek (children) táblára. |
| game\_id | BIGINT | Hivatkozás a játék (games) táblára. |
| created\_at, updated\_at | TIMESTAMP | Laravel időbélyegek automatikusan. |

* Seeders
* Ez a **seeder** három felhasználót hoz létre automatikusan az adatbázisban:

| **Név** | **Email** | **Szerep** | **Jelszó** |
| --- | --- | --- | --- |
| Test User | test@example.com | user (alapértelmezett) | nincs megadva (factory generálja) |
| Admin | admin@example.com | admin | admin123 |
| SuperAdmin | superadmin@example.com | superadmin | superadmin123 |

**Mikor használjuk ezt?**

* Fejlesztéskor: hogy mindig legyen adminisztrátorod és szuperadminod az alkalmazásban belépéshez.
* Teszteléskor: hogy ne kelljen kézzel újra és újra regisztrálni felhasználókat.
* Demókészítéskor: előkészített adatokkal indul az alkalmazás.
* Database

**Adatbázis-tervezet**

**1. users (felhasználók)**

| **Oszlop** | **Típus** | **Leírás** |
| --- | --- | --- |
| id | BIGINT (auto-increment) | Egyedi azonosító |
| name | VARCHAR(255) | Felhasználó neve |
| email | VARCHAR(255) (egyedi) | Felhasználó email címe |
| email\_verified\_at | TIMESTAMP (nullable) | Email megerősítés ideje |
| password | VARCHAR(255) | Jelszó (bcrypt hash-elt) |
| login\_counter | INT (nullable, alapértelmezett 0) | Sikertelen belépési próbálkozások száma |
| banning\_time | TIMESTAMP (nullable) | Kitiltás időpontja |
| role | ENUM('user', 'admin', 'superadmin') | Felhasználói szerepkör |
| remember\_token | VARCHAR(100) (nullable) | "Remember me" token |
| created\_at | TIMESTAMP | Létrehozás ideje |
| updated\_at | TIMESTAMP | Módosítás ideje |

**2. children (gyermekek)**

| **Oszlop** | **Típus** | **Leírás** |
| --- | --- | --- |
| id | BIGINT (auto-increment) | Egyedi azonosító |
| user\_id | BIGINT (külső kulcs: users.id) | Szülő azonosító |
| name | VARCHAR(255) | Gyermek neve |
| age | INT | Gyermek életkora |
| created\_at | TIMESTAMP | Létrehozás ideje |
| updated\_at | TIMESTAMP | Módosítás ideje |

**3. games (játékok)**

| **Oszlop** | **Típus** | **Leírás** |
| --- | --- | --- |
| id | BIGINT (auto-increment) | Egyedi azonosító |
| title | VARCHAR(255) | Játék címe |
| description | TEXT | Játék leírása |
| category | VARCHAR(255) | Játék kategóriája |
| created\_at | TIMESTAMP | Létrehozás ideje |
| updated\_at | TIMESTAMP | Módosítás ideje |

**4. statistics (statisztikák)**

| **Oszlop** | **Típus** | **Leírás** |
| --- | --- | --- |
| id | BIGINT (auto-increment) | Egyedi azonosító |
| child\_id | BIGINT (külső kulcs: children.id) | Gyermek azonosító |
| game\_id | BIGINT (külső kulcs: games.id) | Játék azonosító |
| score | INT | Eredmény / pontszám |
| play\_time | INT | Játékidő másodpercben |
| created\_at | TIMESTAMP | Létrehozás ideje |
| updated\_at | TIMESTAMP | Módosítás ideje |

**5. child\_game (pivot tábla gyermek-játék kapcsolathoz)**

| **Oszlop** | **Típus** | **Leírás** |
| --- | --- | --- |
| id | BIGINT (auto-increment) | Egyedi azonosító |
| child\_id | BIGINT (külső kulcs: children.id) | Gyermek azonosító |
| game\_id | BIGINT (külső kulcs: games.id) | Játék azonosító |
| created\_at | TIMESTAMP | Létrehozás ideje |
| updated\_at | TIMESTAMP | Módosítás ideje |

**6. password\_reset\_tokens (jelszó-visszaállító tokenek)**

| **Oszlop** | **Típus** | **Leírás** |
| --- | --- | --- |
| email | VARCHAR(255) (primary) | Felhasználó email címe |
| token | VARCHAR(255) | Visszaállító token |
| created\_at | TIMESTAMP (nullable) | Létrehozás ideje |

**7. sessions (aktív felhasználói munkamenetek)**

| **Oszlop** | **Típus** | **Leírás** |
| --- | --- | --- |
| id | VARCHAR(255) (primary) | Munkamenet azonosító |
| user\_id | BIGINT (nullable) | Felhasználó azonosító |
| ip\_address | VARCHAR(45) (nullable) | IP cím |
| user\_agent | TEXT (nullable) | Böngésző adatok |
| payload | LONGTEXT | Serializált munkamenet adatok |
| last\_activity | INT | Utolsó aktivitás ideje (timestamp) |

**8. cache, cache\_locks (gyorsítótár kezelő táblák)**

* **cache:** kulcs-érték párok időzített tárolása.
* **cache\_locks:** egyszerű zárolás a cache adatokhoz.

**9. jobs, failed\_jobs, job\_batches (feladatkezelő queue táblák)**

* **jobs:** várakozó feladatok adatai (pl. háttérfeldolgozások).
* **failed\_jobs:** sikertelenül lefutott feladatok logja.
* **job\_batches:** csoportos feladatok kezelése.

**10. personal\_access\_tokens (API tokenek a Sanctum számára)**

| **Oszlop** | **Típus** | **Leírás** |
| --- | --- | --- |
| id | BIGINT (auto-increment) | Egyedi azonosító |
| tokenable\_type | STRING | Model típusa (pl. User) |
| tokenable\_id | BIGINT | Model ID |
| name | STRING | Token név |
| token | STRING (64) | Egyedi token |
| abilities | TEXT (nullable) | Jogosultságok (pl. ["read", "write"]) |
| last\_used\_at | TIMESTAMP (nullable) | Utolsó használat ideje |
| expires\_at | TIMESTAMP (nullable) | Lejárat ideje |
| created\_at | TIMESTAMP | Létrehozás ideje |
| updated\_at | TIMESTAMP | Módosítás ideje |

**Táblák közötti kapcsolatok (Relációk)**

 users.id ⟷ children.user\_id

 children.id ⟷ child\_game.child\_id

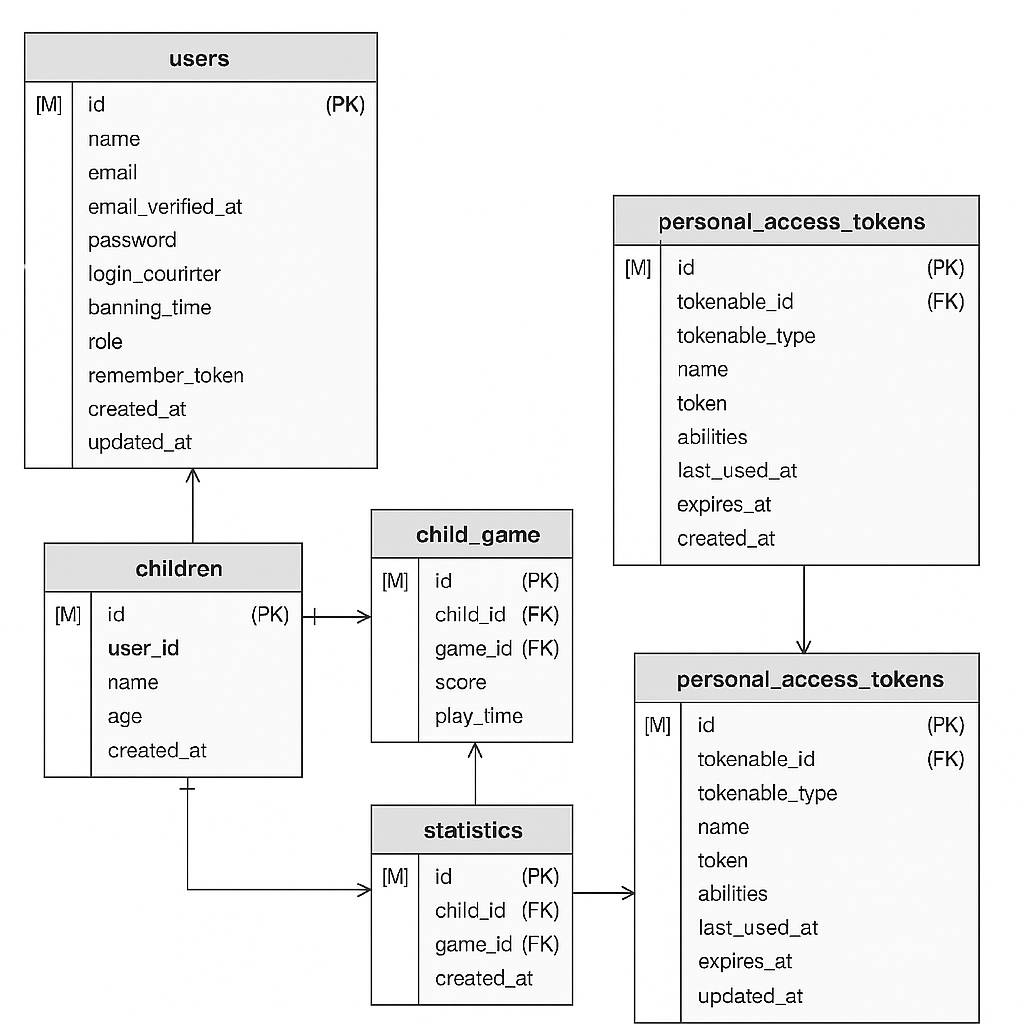
 games.id ⟷ child\_game.game\_id

 children.id ⟷ statistics.child\_id

 games.id ⟷ statistics.game\_id

 users.id ⟷ sessions.user\_id

**Adatbázis modell**

****