Rendszerterv

Online Gallery

Szoftverarchitektúrák (VIAUMA06) tárgy házi feladat

Markovics Gergely (ME7KKE)

Nagy Gellért (A2PD2Q)

Tartalom

[Megvalósítás 3](#_Toc120441275)

[Backend 3](#_Toc120441276)

[DAL réteg 3](#_Toc120441277)

[BLL réteg 9](#_Toc120441278)

[API réteg 10](#_Toc120441279)

[Frontend 10](#_Toc120441280)

[Interceptorok 11](#_Toc120441281)

[Guardok 11](#_Toc120441282)

[Szolgáltatások 11](#_Toc120441283)

[Komponensek 12](#_Toc120441284)

[Telepítési leírás 24](#_Toc120441285)

[A program készítése során felhasznált eszközök 25](#_Toc120441286)

[Összefoglalás 26](#_Toc120441287)

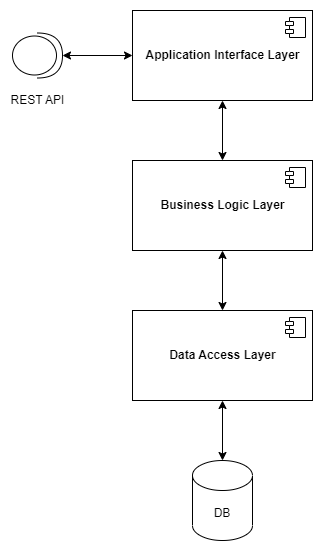
[Továbbfejlesztési lehetőségek 27](#_Toc120441288)

[Hivatkozások 28](#_Toc120441289)

# Megvalósítás

## Backend

Az alkalmazás szerveroldali komponensét .NET 6 platformon implementáljuk. Ez egy cross-platform megoldás, tehát lehetőségünk van a program tetszőleges operációs rendszeren való futtatására. Az alkalmazás a háromrétegű architektúra elvét követi: definiálunk adathozzáférési, üzleti logikai és interfész rétegeket, ezek a solutionben külön projektek. Közvetlen módon mindegyik réteg csak az eggyel alacsonyabb szintűtől függ. A szerver a szolgáltatást egy REST API típusú interfészen keresztül publikálja. Az alábbiakban az egyes rétegek szerepeit részletezzük.



1. Figure A szerver architektúrája

### DAL réteg

#### Entity Framework

Az adatbázis műveletek megvalósításához a .NET Entity Framework keretrendszerét használjuk. Ez egy ORM leképezési réteget biztosít a C# osztályok és az adatbázis táblái között. A lekérdezések megfogalmazása során ezért nem kell sima SQL nyelvet használnunk, elég a C# entitásokon végrehajtani ezeket, az átfordítást az Entity Framework elvégzi számunkra. Lehetőségünk van arra is, hogy a definiált C# osztályok alapján hozzuk létre az adatbázis sémáját (code-first megközelítés). Ekkor a keretrendszer elvégzi a C# és SQL típusok megfeleltetését, az öröklési hierarchiák és a táblák közötti kapcsolatok felderítését. Ez utóbbi névkonvenció alapján történik, ezért fontos, hogy a változók számára megfelelő neveket válasszunk, amennyiben ki akarjuk ezt használni (pl. kulcs esetén Id postfix, lista esetén {entitásnév}s). Ha a modell osztályokban leírt sémát szeretnénk átültetni az adatbázisba, akkor azt egy migráció létrehozásával, majd annak alkalmazásával tehetjük meg. A séma esetleges hibáit (pl. nem felderíthető kapcsolat, ciklikus kaszkád törlés) is ekkor hozza tudomásunkra a rendszer. Amennyiben az entitásokat manuálisan (is) szeretnénk konfigurálni, akkor ezt megtehetjük többek között az adatbázist reprezentáló osztály OnModelCreating metódusában.

#### Adatbázis séma

Az alkalmazás adatbázis sémája az alábbi képen látható. A továbbiakban ismertetjük azokat az entitásokat, amelyeket az alkalmazás használ (a felhasználókezelést megvalósító AspNetCore.Identity könytár adattáblái közül nem az összeset használjuk fel).



2. Figure Az alkalmazás adatbázis sémája

#### Felhasználó entitás

**Célja:** A regisztrált felhasználókat tároljuk ebben a táblában.

**Leképezése a kódban:** Az ApplicationUser osztály

**Leképezése az adatbázisban:** Az AspNetUsers tábla 1-1 sora

**Tulajdonságai:**

|  |  |  |
| --- | --- | --- |
| **C# mezőnév** | **C# adattípus** | **SQL Server adattípus** |
| Id | Guid | UNIQUEIDENTIFIER PRIMARY KEY |
| FirstName | string | NVARCHAR (MAX) |
| LastName | string | NVARCHAR (MAX) |
| UserName | string | NVARCHAR (256) |
| PasswordHash | string | NVARCHAR (MAX) |
| Email | string | NVARCHAR (256) |
| … | … | … |

A fenti táblázat csak az alkalmazás szempontjából fontos tulajdonságokat sorolja fel.

* Az Id mező tárolja az entitás egyedi azonosítóját. Ez alapján lehet az adatbázisban az entitást azonosítani.
* A FirstName mező tartalmazza a felhasználó keresztnevét.
* A LastName mező tartalmazza a felhasználó vezetéknevét.
* A UserName mező tartalmazza a felhasználó által választott felhasználónevet.
* A PasswordHash mező tartalmazza a fiókhoz tartozó jelszó hashét.
* A felhasználó email címét az Email mező tárolja.

#### Szerep entitás

**Célja:** A felhasználók lehetséges szerepköreit rögzíti. A jogosultságkezeléshez használatos.

**Leképezése a kódban:** Az ApplicationRole osztály

**Leképezése az adatbázisban:** Az AspNetRoles tábla 1-1 sora

**Tulajdonságai:**

|  |  |  |
| --- | --- | --- |
| **C# mezőnév** | **C# adattípus** | **SQL Server adattípus** |
| Id | Guid | UNIQUEIDENTIFIER PRIMARY KEY |
| Description | string | NVARCHAR (MAX) |
| Name | string | NVARCHAR (256) |
| NormalizedName | string | NVARCHAR (256) |
| ConcurrencyStamp | string | NVARCHAR (MAX) |

* Az Id mező tárolja az entitás egyedi azonosítóját. Ez alapján lehet az adatbázisban az entitást azonosítani.
* A Description mező egy rövid leírást tartalmaz az adott szerepkörről.
* A szerepkör nevét a Name mező tartalmazza, ennek segítségével hivatkozunk rá a programban.
* A NormalizedName mező a szerepkör nevét tartalmazza csupa nagybetűvel írva.
* Az esetleges konkurrencia problémákat hivatott megelőzni a ConcurrencyStamp mező.

#### Felhasználó-szerep entitás

**Célja:** Kapcsolótábla, amire a felhasználók és a szerepkörök N-N kapcsolata miatt van szükség. Összerendeli a felhasználókat a hozzá tartozó szerepkörökkel.

**Leképezése a kódban:** Nem hivatkozunk rá expliciten, navigációs tulajdonságként érhető el.

**Leképezése az adatbázisban:** Az AspNetUserRoles tábla 1-1 sora

**Tulajdonságai:**

|  |  |  |
| --- | --- | --- |
| **C# mezőnév** | **C# adattípus** | **SQL Server adattípus** |
| UserId | Guid | UNIQUEIDENTIFIER FOREIGN KEY |
| RoleId | Guid | UNIQUEIDENTIFIER FOREIGN KEY |

* A UserId mező tárolja, hogy mely felhasználót kapcsoljuk hozzá az adott szerepkörhöz.
* A RoleId mező tárolja, hogy mely szerepkört kapcsoljuk hozzá az adott felhasználóhoz.

#### Album entitás

**Célja:** A felhasználók által létrehozott albumokat tárolja.

**Leképezése a kódban:** Az Album osztály

**Leképezése az adatbázisban:** Az Albums tábla 1-1 sora

**Tulajdonságai:**

|  |  |  |
| --- | --- | --- |
| **C# mezőnév** | **C# adattípus** | **SQL Server adattípus** |
| Id | Guid | UNIQUEIDENTIFIER PRIMARY KEY |
| Creator | ApplicationUser | UNIQUEIDENTIFIER FOREIGN KEY |
| Name | string | NVARCHAR (MAX) |
| IsPrivate | bool | BIT |
| LikeCount | int | INT |

* Az Id mező tárolja az entitás egyedi azonosítóját. Ez alapján lehet az adatbázisban az entitást azonosítani.
* A Creator navigációs tulajdonság hivatkozik a létrehozó felhasználó entitásra. Ez adatbázis szinten egy külső kulcs formájában jelenik meg.
* A Name mező tartalmazza az album felhasználó által megadott nevét.
* Az IsPrivate bool flag jelzi, hogy az adott album privát-e.
* A LikeCount mezőben tároljuk, hogy hány ember like-olta az albumot.

#### Kép entitás

**Célja:** A felhasználók által feltöltött egyes képek adatait tároljuk itt.

**Leképezése a kódban:** A Picture osztály

**Leképezése az adatbázisban:** A Pictures tábla 1-1 sora

**Tulajdonságai:**

|  |  |  |
| --- | --- | --- |
| **C# mezőnév** | **C# adattípus** | **SQL Server adattípus** |
| Id | Guid | UNIQUEIDENTIFIER PRIMARY KEY |
| Album | Album | UNIQUEIDENTIFIER FOREIGN KEY |
| PhysicalPath | string | NVARCHAR (MAX) |
| DisplayPath | string | NVARCHAR (MAX) |
| Size | double | FLOAT (53) |
| FileExtension | string | NVARCHAR (MAX) |

* Az Id mező tárolja az entitás egyedi azonosítóját. Ez alapján lehet az adatbázisban az entitást azonosítani.
* A Creator navigációs tulajdonság hivatkozik a tartalmazó albumra. Ez adatbázis szinten egy külső kulcs formájában jelenik meg.
* A kép abszolút fizikai elérési útvonalát a PhysicalPath tulajdonság tárolja.
* A DisplayPath mező a kép relatív útvonalát tárolja, amit az alap URL mögé illesztve elérhetjük azt a szerverről.
* A Size tulajdonság a kép méretét tárolja MB-ban.
* A FileExtension a kép formátumát rögzíti.

#### Kedvenc album entitás

**Célja:** A felhasználók és a kedvenc albumok összerendelő táblázata.

**Leképezése a kódban:** FavoritedBy navigációs tulajdonság az Album osztályban és FavoritedAlbums navigációs osztály az ApplicationUser osztályban

**Leképezése az adatbázisban:** A Favorite tábla 1-1 sora

**Tulajdonságai:**

|  |  |  |
| --- | --- | --- |
| **C# mezőnév** | **C# adattípus** | **SQL Server adattípus** |
| FavoritedAlbumsId | Album | UNIQUEIDENTIFIER FOREIGN KEY |
| FavoritedById | ApplicationUser | UNIQUEIDENTIFIER FOREIGN KEY |

* A FavoritedAlbumsId mező tárolja, hogy mely albumot kapcsoljuk hozzá az adott felhasználóhoz. Ez adatbázis szinten egy külső kulcs formájában jelenik meg.
* A FavoritedById mező tárolja, hogy mely felhasználót kapcsoljuk hozzá az adott albumhoz. Ez adatbázis szinten egy külső kulcs formájában jelenik meg.

#### Repository és Unit of work

Az entitásokkal való műveletek során vannak olyanok, amiket szinte mindig definiálni kell. Ilyen a valamilyen feltétel alapján való keresés, törlés, módosítás és létrehozás. Azért, hogy ezeket a műveleteket csak egyszer definiáljuk, létrehozunk egy generikus repository osztályt, és ezeket a közös műveleteket itt implementáljuk.

Ha egy művelet végrehajtásához sok különböző entitással kell dolgozunk, akkor az ezekkel paraméterezett repository osztályokat egyesével kéne létrehoznunk. Annak érdekében, hogy ezt megspóroljuk és az adatelérést absztraháljuk, létrehozunk egy Unit of work osztályt, ami tartalmazza az összes entitáshoz tartozó repository osztályt. Ekkor bármilyen adatbázis művelet végrehajtása esetén elég csak ebből az osztályból létrehozni egy példányt, így elérve az összes definiált entitást.

#### Fájlok tárolása

A képek tárolásához az operációs rendszer fájlrendszerét használjuk. A pontos fizikai útvonal az appsettings.json fájlban található értéktől függ. A feltöltött fájlokat felhasználók szerint rendezzük: egy adott kép a feltöltő azonosítójával elnevezett mappába kerül.

Ahhoz, hogy a webszerver kiszolgálja ezeket a kéréseket, el kell helyeznünk a megfelelő middleware-t a HTTP kérés pipelineban. Ennek a konfigurálásánál megadunk egy fizikai útvonalat és egy kérés útvonalat. Ezután a kérés útvonalához hozzáillesztve a fájl elérésének relatív útvonalát az adott mappából kapunk egy arra mutató linket.

#### Konfiguráció

Az alkalmazás futása során szükségünk lehet bizonyos konfigurációs értékekre, amik bizonyos beállításokat tartalmaznak (pl. mi legyen a maximális feltölthető fájlméret). Ezeket az appsettings.json fájlban tároljuk, azonban mi szeretnénk ezekhez az értékekhez plusz üzleti logikát rendelni, ezért bevezetünk egy konfigurációs szolgáltatást. Az alkalmazás egyéb részei ezen keresztül férnek hozzá ezekhez a konfigurációs értékekhez.

### BLL réteg

#### Mediator minta

Az alkalmazás az üzleti logikai műveletek elvégzéséhez a mediator mintát használja, a MediatR csomag használatával. Ennek lényege, hogy az API réteg nem a BLL réteg szolgáltatásaival kommunikál közvetlenül, hanem egy IMediator típusú objektummal, és ennek az objektumnak a feladata, hogy a kérést a megfelelő szolgáltatás megkapja és végrehajtsa.

Ennek megfelelően a BLL rétegban definiálunk parancsokat (command) és kéréseket (query), amiket szétválasztunk az egyes domain entitásokra tekintve (jelen esetben felhasználó és album). A két típus között az a különbség, hogy a command valamilyen adatmódosítással járó műveletet valósít meg, míg a query csak adatlekérést. Ezen az objektumok tartalmazzák mindazt az adatot, ami az adott művelet végrehajtásához szükséges. Ezeket az objektumokat kapják meg a handler osztályok, ahol a különböző típusú parancs és kérés osztályokat kezelő üzleti logikát implementáljuk.

#### Automapper

Az entitás és a kommunikációt végrehajtó osztályok közötti átalakítást nagyon sokszor elvégezzük, ezért célszerű ezt a logikát központosítani. Az egyik legnépszerűbb megoldás erre az Automapper könyvtár, ami a megfeleltetési logika széles palettáját támogatja, kezdve az egyszerű név szerinti megfeleltetési logikától a külön osztályokban definiált, tulajdonság vagy típus szerinti átalakításokig. A különböző domain entitásokhoz külön ún. profil osztályokat hozunk létre, amiknek konstruktorában megadhatjuk az adott entitás átalakítási logikáit. Amennyiben egy adott tulajdonsághoz vagy típushoz komplex műveleteket kell végezni, akkor célszerű az adott logikát kiszervezni egy külön osztályba (ahol akár elkérhetünk szolgáltatásokat a DI konténtertől), majd ezt az osztályt behivatkozni az átalakítási logikában.

#### Validálás

Az alkalmazás futása során különböző validációkra van szükségünk (pl. egy album törlését csak a létrehozó felhasználó vagy egy admin teheti meg, nem bármely bejelentkezett felhasználó). Ezeket célszerű kiszervezni más osztályokba, hogy a handler osztály a művelet végrehajtására tudjon koncentrálni. Ennek érdekében definiálunk egy IValidator interfészt, aminek egy bool visszatérésű Validate metódusa van, a különböző implementációk pedig ezt fogják megvalósítani. Annak érdekében, hogy ezek között kapcsolatokat is definiálhassunk (pl. vagy, és), létrehozunk “összekötő” osztályokat, amik kapnak két IValidator típusú objektumot, és a Validate metódusukban a megfelelő logikai művelettel kapcsolják azokat össze. Ennek segítségével egy IValidator típusú objektummal is leírhatjuk az összes, adott művelethez szükséges validációs kritériumot.

### API réteg

#### Kontrollerek

Az alkalmazás REST API-n keresztül érhető el. A beérkező kéréseket kontroller osztályok fogadják, amik egy adott entitáson végezhető műveleteket foglalják egy csoportba (pl. album létrehozása, törlése, módosítása). A kontroller metódusait ellátjuk annotációkkal, amely megfelelteti azt egy HTTP ige és útvonal kombinációjával. Ezen osztályok csak az Imediator objektummal keresztül kommunikálnak a BLL komponenseivel, ezért minden hívásnak nagyon hasonló annak kezelése: létrehozzuk a kéréshez tartozó kérés/parancs objektumot, annak átadva minden adatot amire szüksége van, majd ezen létrehozott objektumot átadjuk a mediatornak, aki a megfelelő handler metódust hívja meg ennek hatására.

Az egyes kontroller osztályokra vagy annak metódusaira elhelyezzük az Authorize vagy AllowAnonymous attribútumot attól függően, hogy az adott művelethez be kell-e jelentkezve lennie az adott felhasználónak. Az Authorize attribútumnak megadjuk azt is, hogy milyen szerepkörben kell lennie a bejelentkezett felhasználónak (sima, admin, bármilyen) a sikeres műveletvégzéshez.

#### Autentikálás

Az autentikáláshoz IdentityServert használtunk, ami JWT alapon működik. Ez a könyvtár rengeteg különféle, OAuth 2.0 kompatibilis autentikációs folyamatot tartalmaz különféle igényekhez. Mivel nekünk csak jelszó alapú bejelentkezés kell, ezért a Resource owner password folyamatot valósítjuk meg. Ennek során egy sikeres felhasználónév (vagy email) és jelszó páros megadásakor a felhasználó kap egy JWT-t, ami tartalmazza az ő adatait, többek között az id-t, a nevét és szerepkörét. A kliens a szerepkör alapján jeleníthet meg vagy rejthet el bizonyos funkciókat a felhasználói felületen, illetve a szerver ezt a tokent használhatja az azonosításhoz és validáláshoz.

Az alkalmazás elindításakor az appsettings.json fájlban található szerepkörökkel és felhasználó fiókokkal feltöltjük az adatbázist, így adhatunk hozzá új adminisztrátori fiókokat. A regisztrált felhasználók mindig normál szerepkörrel rendelkeznek.

#### Hibakezelés

A hibakezelést a Helland.Middleware.ProblemDetails könyvtár segítségével valósítottuk meg. Ennek segítségével elhelyezünk egy middlewaret a HTTP kérés pipelineban, ahol megadhatjuk az alkalmazás futása során előforduló kivétel típusokhoz tartozó válasz kódot és üzenetet.

## Frontend

A szoftver felhasználóval való kommunikációjáért egy Angular alapú webalkalmazás szerepel. Ez az alkalmazás továbbítja és kéri le az adatokat a szerver felé és felől. Az alkalmazás JWT segítségével jegyzi meg a belépett felhasználót, és későbbiekben azzal azonosítja a hívások alatt.

A dokumentáció során bemutatásra kerülnek az alkalmazás főbb eszközei és komponensei, kezdve a működést elősegítő eszközökkel.

### Interceptorok

Angularban az interceptorok segítségével tudjuk elkapni a hívásokat, és esetlegesen továbbítani a fogadónak. Szerepe, hogy a hívásokon implementálhatunk velük kiegészítő funkciókat.

A jelen implementációban egyetlen interceptor használunk, az AuthInterceptor osztályt. Ennek a szerepe, hogy az eltárolt tokent elhelyezze a hívások fejlécében, így megspóroljuk azt a lépést, hogy egyesével felrakjuk minden hívásnál.

Emellett, ha elkapott egy hibaüzenetet, akkor arról értesíti a felhasználót egy snackbarban.

### Guardok

A guardok olyan elemei az alkalmazásban, amik segítségével szabályozni tudjuk, hogy a felhasználó melyik útvonalat és milyen módon érheti el. Ennek tudjuk benne megszabni a feltételeket.

Az alkalmazásban kétféle guard van implementálva:

* AuthGuard: Ennek a guardnak a szerepe, hogy az olyan útvonalak meglátogatása esetén, ami felhasználó belépését igényli, csak belépett felhasználók látogathassák meg. Így, ha nem talál eltárolt tokent, akkor a felhasználót átirányítja a belépési oldalra, hogy ott kezdje.
* LoginGuard: Login felület “védelmére” szolgál. Ha a felhasználó bejelentkezett ezelőtt, akkor az alapértelmezett login felület nem igényli, hogy lépjen be újra, helyette átirányítja a felhasználónak a profil felületére.

### Szolgáltatások

A szolgáltatások támogatják a komponensek működését kiegészítő logikával, változók kezelésével vagy szerver felé hívások indításával.

Az alkalmazás számos szolgáltatást tartalmaz, mindegyik egy-egy funkció megvalósítására specializálódik:

* UserService: felhasználóhoz köthető funkciók és szerverhívások kezelése
* AlbumService: Galériákhoz köthető szerverhívások indítása
* TokenService: JWT eltárolására és olvasására specializálódik
* SidebarService: navigációs sáv kezelését valósítja meg
* SnackService: snackbarok megnyitását lehet vele elindítani
* ConfirmService: Megerősítési dialógust lehet vele kezelni
* ConfirmAlbumService: Megerősítési dialógust és kimenetét kezeli a szerverhívások megerősítésére
* LoadingService: Töltési réteg bekapcsolását lehet vele kezelni
* PreviewService: Képek előnézetét mutató komponenst lehet bekapcsolni

Számos szolgáltatást lehet számos komponens működtetésénél használni, azoknak a vékonykliens szintű logikájának megvalósításai.

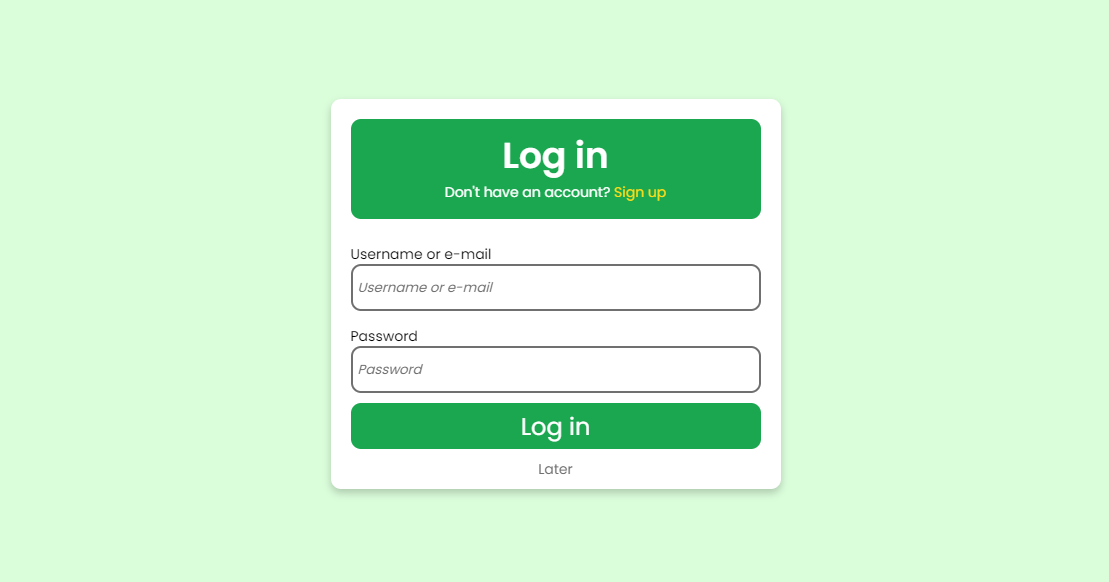
### Komponensek

Az alkalmazás legfontosabb egységei. Az egyes komponensek felelősek a felhasználói felület egészéért vagy egyes alrészéért, komponensek egymásba vagy egymás fölé ágyazásával. Ezek az egységek felelősek a felhasználói élmény biztosításáért, és a felhasználó megfelelő irányításáért.

Egy komponensnek 3 része van. Egy typescript alapú logikai háttér, ami változókat kezel és továbbítja a kéréseket a megfelelő szolgáltatások felé. A felhasználói felület innen kapja közvetlenül a statikus vagy nem statikus adatokat, amiket meg kell jelenítenie. Második az scss stílusleíró nyelvre épülő fájl, amiben a komponens szintű stílusok kerülnek eltárolásra. Ezek a stílusok a globális stílusok felülírására szolgálnak, amik cask az adott komponensre vonatkoznak. Harmadik a html fájl, ami meghatározza a képernyőre kiírandó elemeket. Itt kerülnek meghivatkozásra a typescript fájlban megtalálható változók és metódusok.

Ezután az egyes komponensek felsorolása és működésének magyarázata található. Egyes komponensek önmagukban állnak, mások több más komponenst tartalmaznak magukban.

#### LoginComponent



3. Figure Login komponens

Felhasználó ezzel a komponenssel találkozik a weboldalra látogatáskor. Itt tudja megadni a szerver által eltárolt belépési adatait, amivel tovább tud lépni a személyre szabott felületére. Ha a felhasználónak még nincs regisztrációja, akkor a “Sign up” feliratra kattintva át tud lépni a regisztrációs felületre, vagy a “Later” feliratra kattintva tovább tud lépni. Ilyenkor korlátozott jogokkal, anoním módon tud a galériák között böngészni.

A felhasználó be tud lépni a felhasználónevével, vagy a megadott e-mail címével. A biztonságosan eltárolt jelszavával együtt. Ha a megadott adatai nem voltak megfelelőek, akkor egy snackbar figyelmezteti róla felhasználót. Ha a felhasználó belépett már ebből a böngészőből az adott időn belül, akkor ezt a lépést ki is hagyhatja.

#### RegisterComponent

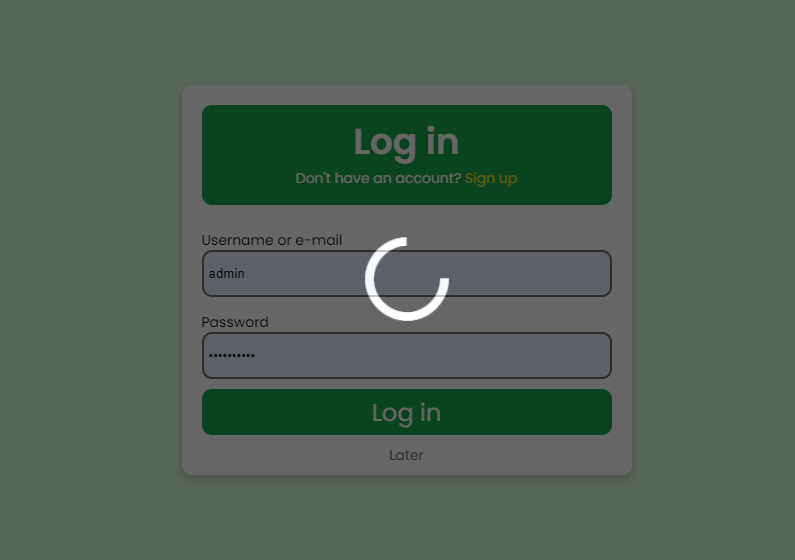


4. Figure Register komponens

Ez a komponens felel a felhasználók létrehozásáért, itt tudják megadni a létrehozandó személyes adatokat. A felhasználónévnek és e-mail címnek egyedinek kell lennie a többi felhasználóhoz képest, amiről a szerver értesíti a felhasználót hiba esetén snackbar segítségével.

A regisztrációs gomb csakk akkor fog aktiválódni, ha minden mező ki lett töltve, és az e-mail formátuma megfelelő. A gomb megnyomása után elküldi a kliens a szervernek a létrehozandó felhasználót, és utána rögtön be is lehet vele lépni a kitöltött személyes adatokkal együtt.

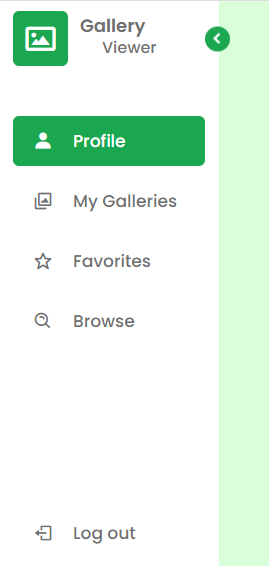
#### LoadingComponent



5. Figure Loading komponens

Egy segédkomponens, ami lezárja a felhasználói felületet, hogy a kliens választ kapjon valamilyen szerver felé intézett hívásra, ezáltal elkerülhetünk esetleges mellékhatásokat, ha a felhasználó nem várja meg, hogy válasz érkezzen.

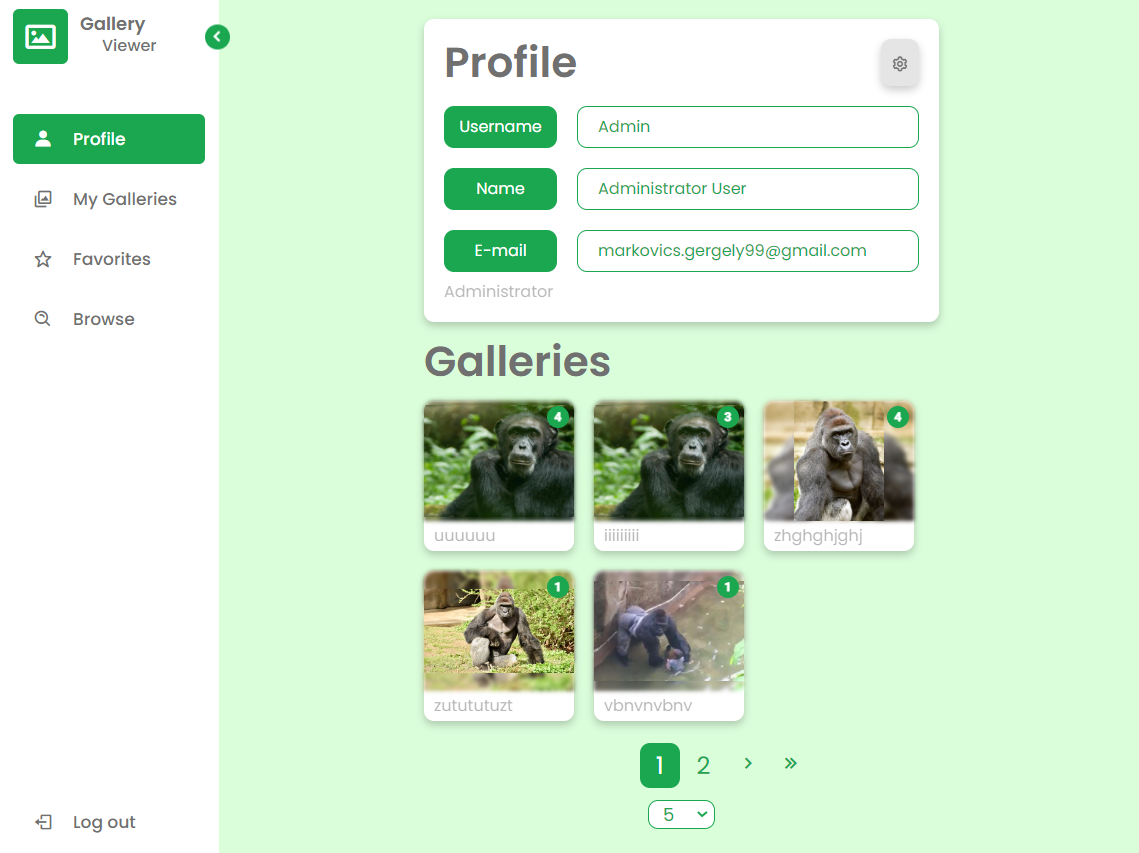
#### SidebarComponent



6. Figure Sidebar komponens

A webalkalmazás navigációs komponense. A felhasználók főleg ennek a komponensnek a használatával tudnak az oldalak között váltani, vagy a felhasználójukból kilépni. Megemlítendő, hogy anoním felhasználóval értelemszerűen egyes funkciók nem érhetőek el, egyedül a galériák közötti böngészés lehetséges.

#### ProfileComponent



7. Figure Profile komponens

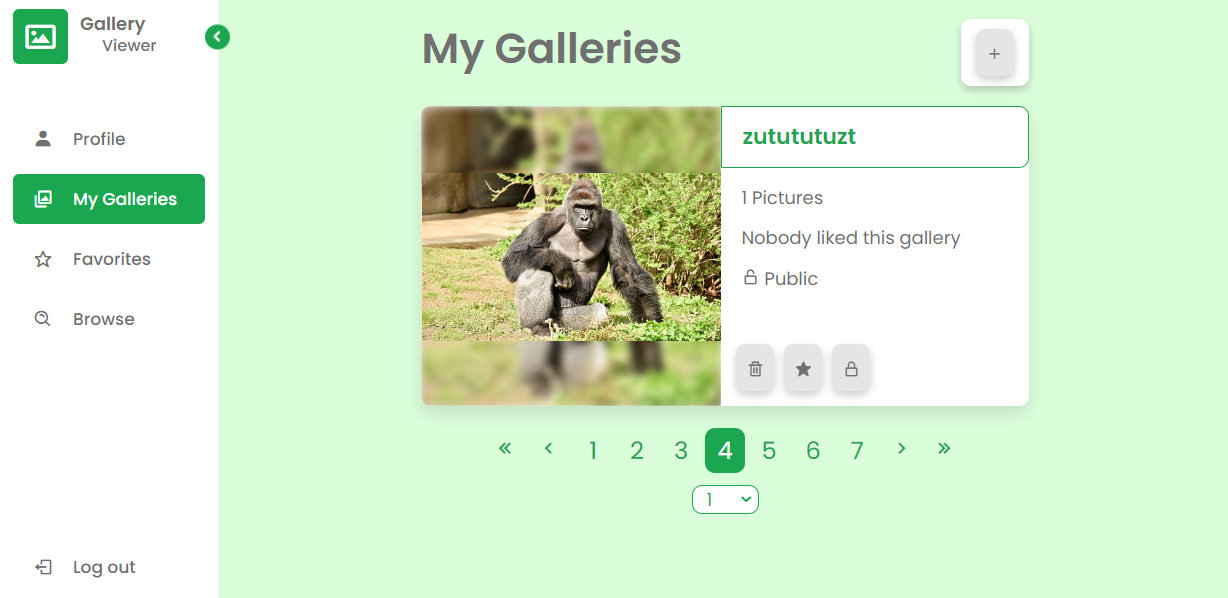
Ez a komponens valósítja meg a felhasználó profilját. Meg van jelenítve a regisztráció során megadott adatai, mint a felhasználóneve, neve és e-mail címe. Emellett megjeleníti azt is, ha admin jogokkal rendelkező a felhasználó.

A felhasználó adatai alatt gyorsgombok találhatóak. Itt sorakoznak fel a felhasználó galériái. A galériákból meg van jelenítve egy kép belőlük, a nevük, és egy kis ikonban, hogy hány darab kép van a galériában. Ezekre kattintva eljuthatunk rögtön az adott galéria részleteit mutató oldalra.

Hogy sok galéria esetén ne lepje el az oldalt gombokkal az alkalmazás, be van vezetve egy pager komponens minden oldalt a listák megjelenítésénél. Itt a listát tudjuk lapozni gombok segítségével, vagy állíthatjuk az alsó választó segítségével az egy oldalon megjelenítendő elemek számát.

A pageren tudunk kattintani a konkrét megjelenő oldalszámokra, vagy a kétoldalt megjelenő gombokkal vagy egyesével oldalra, vagy a lista végére-elejére tudunk ugrani.

#### MyGalleriesComponent



8. Figure My galleries komponens

Ez a komponens felel a felhasználó galériáinak listázására. Itt is ugyanúgy lapozva lehet megtekinteni az egyes vagy többes galériákat. A felhasználó a job felső sarokban tlaálható plusz gomb segítségével tudja megnyitni a galéria létrehozó dialógust, amiben kitöltve az adatokat egy új galéria jelenik meg a listáján.

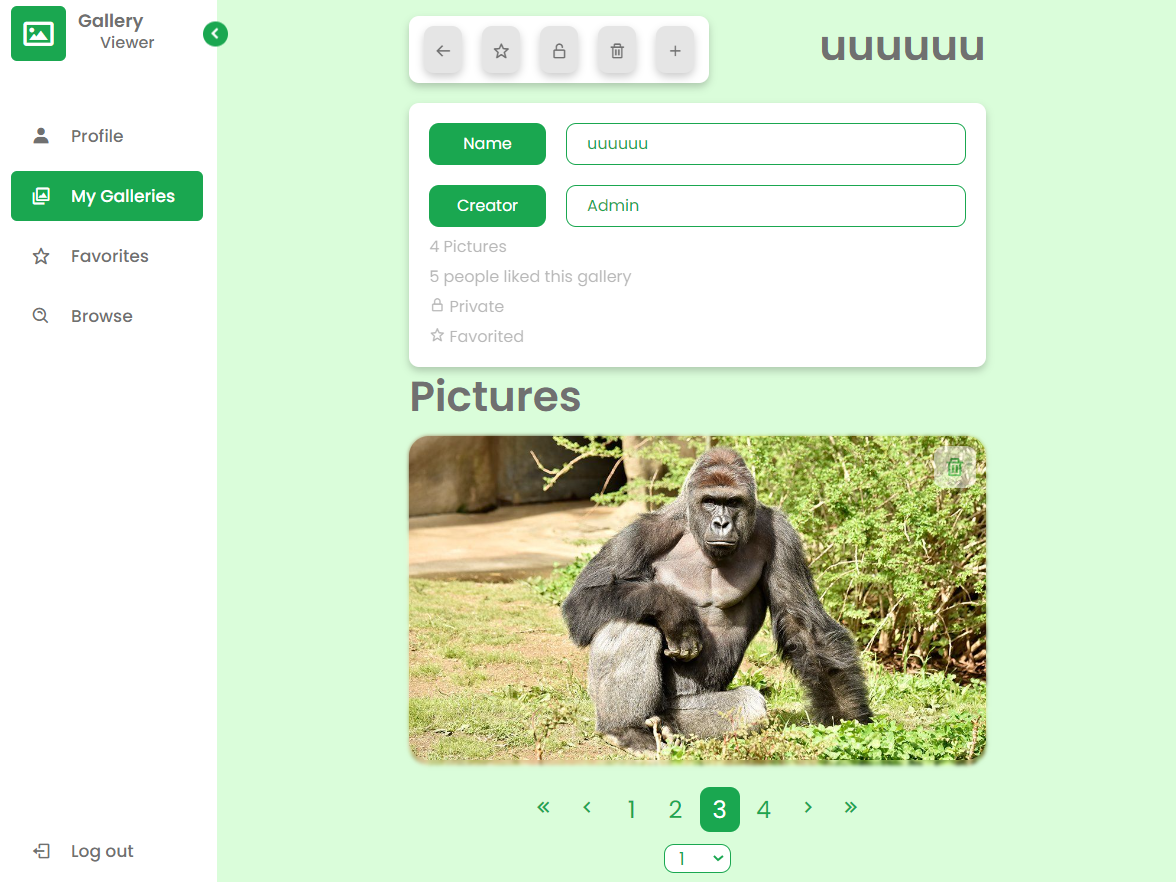
A felhasználói élmény növelésére az egyes galériákon gyorsgombok lettek elhelyezve a 3 fő művelet elvégzésére, hogy ne kelljen a felhasználónak egyesével megnyitnia a galériákat a szerkesztésükhöz.

Ez a három funkció a galéria törlése, kedvencek közé rakása vagy eltávolítása, és a galéria nyilvánosságának változtatása.

Az egyes galériákon meg vannak jelenítve a fő információi az előnézet képek mellett. Látni lehet, hogy hány kép található a galériában. Látni lehet, hogy hány embernek tetszett vagy rakta a kedvencei közé. És látni lehet, hogy a galéria privát, vagy publikus a nyilvánosság felé.

Ha nem a gyorsgombokra, hanem magukra a galériákra kattintunk, akkor eljutunk a galériák részletes nézetére.

#### GalleryComponent



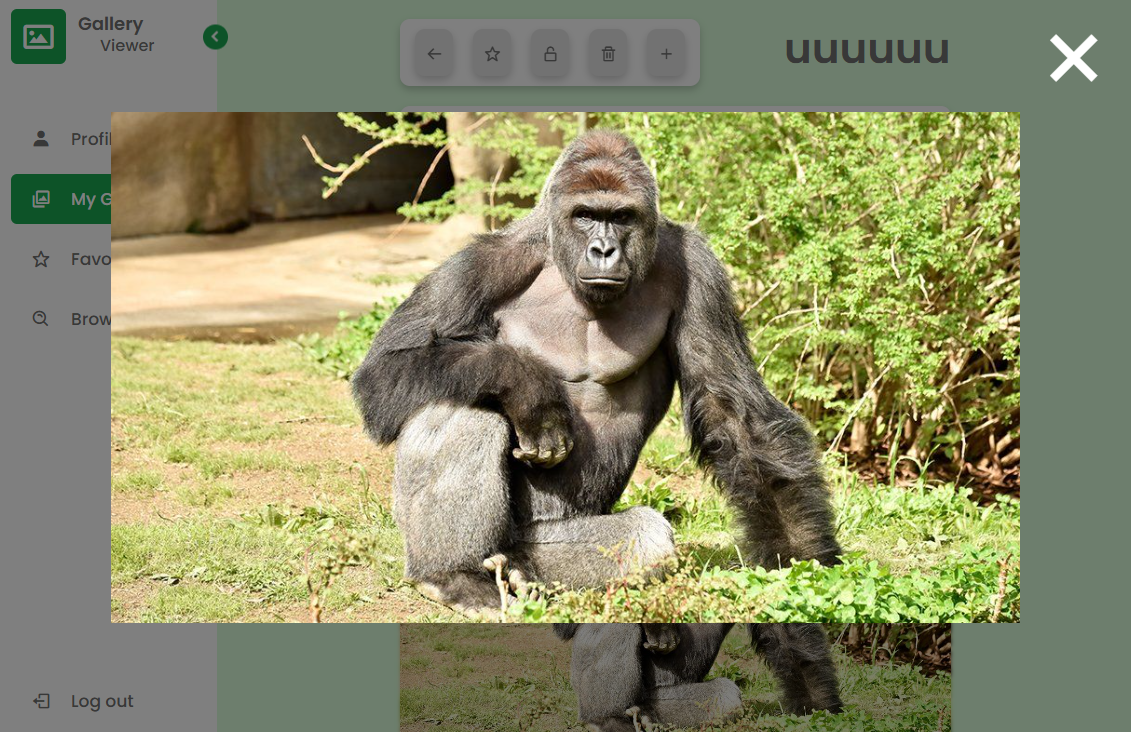
9. Figure Gallery komponens

Ez a segédkomponens beültet egy részletmegtekintési oldalt a felületre. Itt megtekinthetőek a galéria részletes adatai a nevével és létrehozójával együtt.

Emellett itt látjuk felsorolva pager segítségével a galériában található képeket. Az egyes képek jobb felső sarkában található gombbal tudjuk ezeket a képeket törölni.

A részletes módosítási műveletek a bal felső sávban találhatóak. Itt tudunk visszalépni a galéria listára, és a galérián található 3 gyorsművelet elvégzése is itt lehetséges. Az utolsó gomb további képek hozzáadására szolgál, ahol a helyi tárhelyről tudunk kiválasztani képformátumú fájlokat, amiket felküldünk utána a szervernek.

#### PreviewComponent

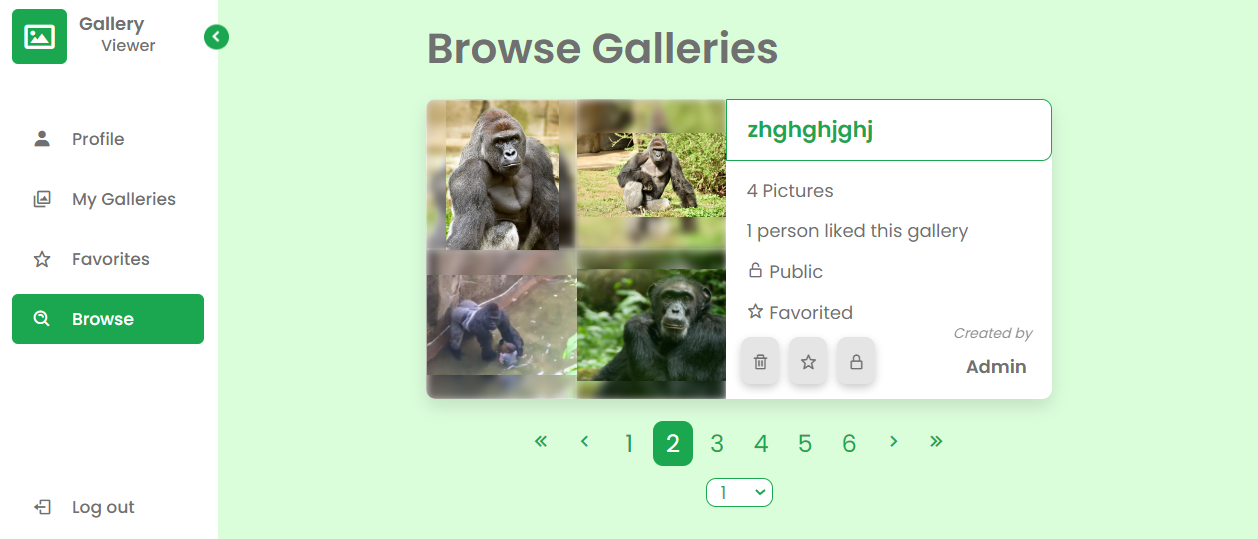


10. Figure Preview komponens

Ennek a komponensnek a segítségével meg tudjuk vizsgálni közelebbről a képeket. Kilépni a job felső sarokban található “X” segítségével tudunk.

Ezt a komponenst a GalleryComponentben felsorolt képekre kattintva tudjuk előhozni, és az ott kiválasztott képet nagyban megtekinteni.

#### BrowseComponent



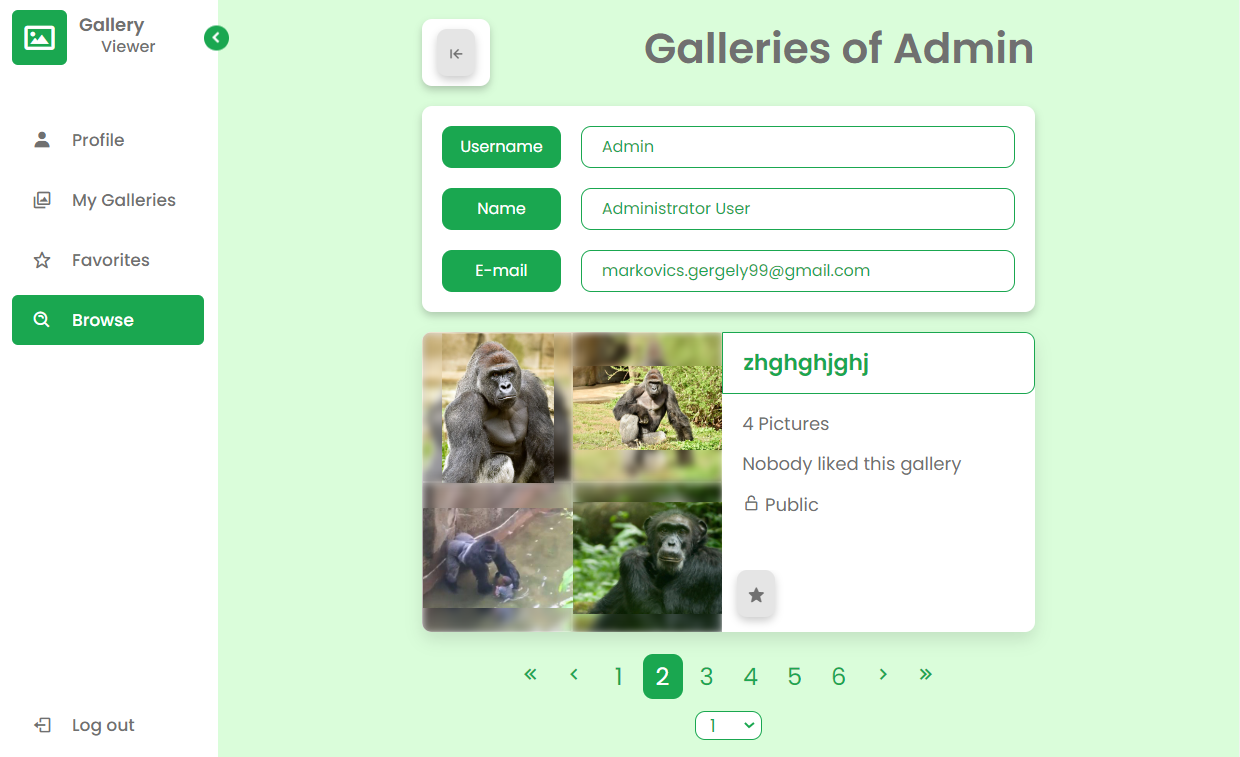
11. Figure Browse komponens

Az alkalmazás fő komponense, itt tudjuk böngészni a felhasználók galériáit. Csak a publikus galériák jelennek meg, kivéve, ha a felhasználó saját galériája, akkor mindkét fajta. Ez a fajta felsorolás abban különleges, hogy anoním felhasználóként is megtekinthető. Ekkor a szokásos gyorsgombok helyett egyetlen szív ikon jelenik meg a galéria tetszésének mutatására. Azért, hogy ne lehessen visszaélni a gomb túlzott használatával, 5 másodpercig minden gombnyomás után letiltja.

A saját galériától még abban különbözik, hogy itt fel van tüntetve a galéria szerzője is. A szerzőre rá lehet kattintani, amikor az alkalmazás átvált a szerző galériáinak a listájára.

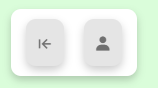
A saját galériánál megtekinthető részletek felület itt is működik a galériára kattintva. Annyi a különbség, hogy itt, ha saját galériára kattintunk akkor átirányít a saját galériáink közül való megtekintésre, ha másnak a galériáját tekintjük meg, akkor egy korlátozott jogokkal rendelkező felületre lépünk. Tehát más galériáját nem tudjuk törölni, ha nincs rá jogunk stb.

#### UserGalleryComponent



12. Figure User Gallery komponens

A korábban említett szerző szerint megjelenített felület. Ugyanúgy működik, mint a browse felület, csak felül meg van jelenítve a kiválasztott felhasználó alap adatai, és alatta csak az ő publikus galériái jelennek meg. Ezen a felületen, ha a bejelentkezett felhasználó admin jogokkal rendelkezik, akkor más felhasználókat is admin jogra tud emelni a felül megjelenített navigációs sávon található ikon segítségével.

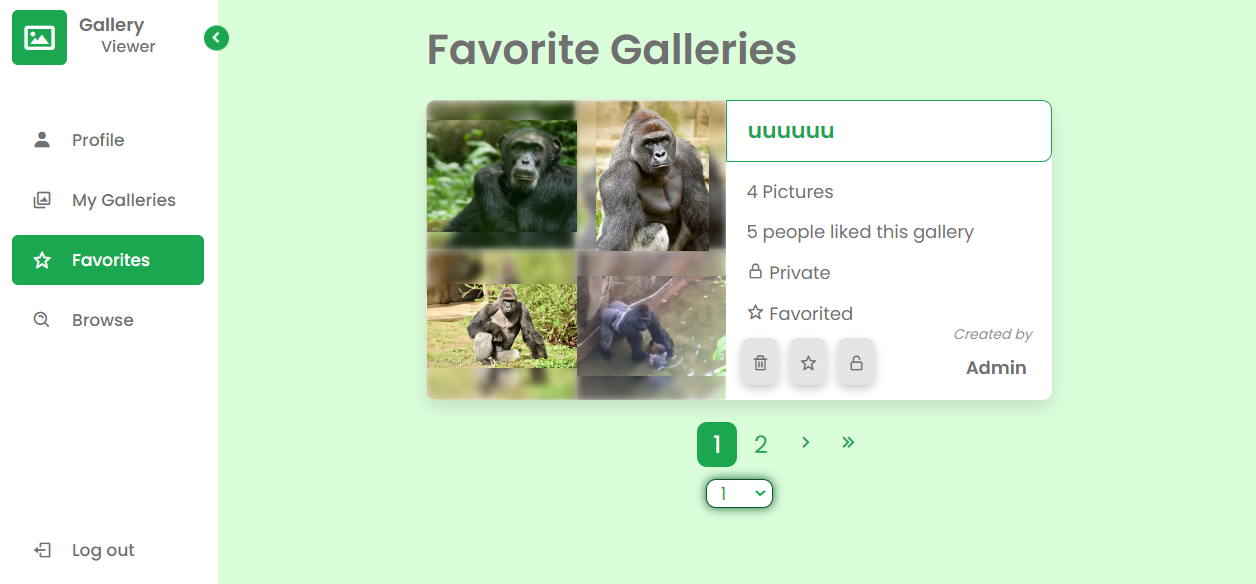
Admin jogokkal más felhasználók galériáit is tudjuk törölni. Ezt az is jelzi, hogy admin felhasználóval itt a kedvencekbe rakás mellett a törlés ikon is megjelenik a gyorsgombok között.

Emellett a kényelem kedvéért minden felhasználónak kétféle visszagomb jelenik meg. Egyik visszavisz a kiválasztott felhasználó oldalára, a másik pedig visszavisz teljesen a browse felületre.

13. Figure User Browse navigációk

14. Figure User browse részletek navigációi

#### FavoritesComponent



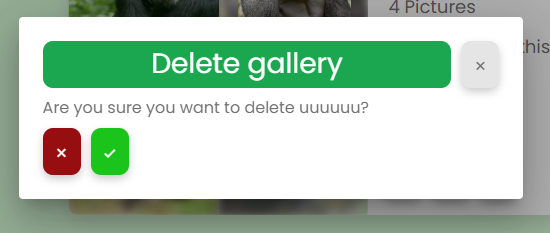
15. Figure Favorites komponens

Ez a komponens sokban hasonlít a browse felületre. Itt a felhasználó által kedvelt galériák kerülnek felsorolásra, így értelemszerűen csak belépett felhasználóknak érhető el. A felhasználói élmény kedvéért a megfelelő jogokkal itt is felkerültek a gyorsgombok a galériákra.

#### Dialógusok

Az alkalmazást Angular modal dialógusok is segítik. Elsődleges feladatuk, hogy feldolgozzanak felhasználói interakciókat, és azt továbbítsák az adott megnyitott komponensnek. Az alkalmazás 2 dialógus komponenst tartalmaz.

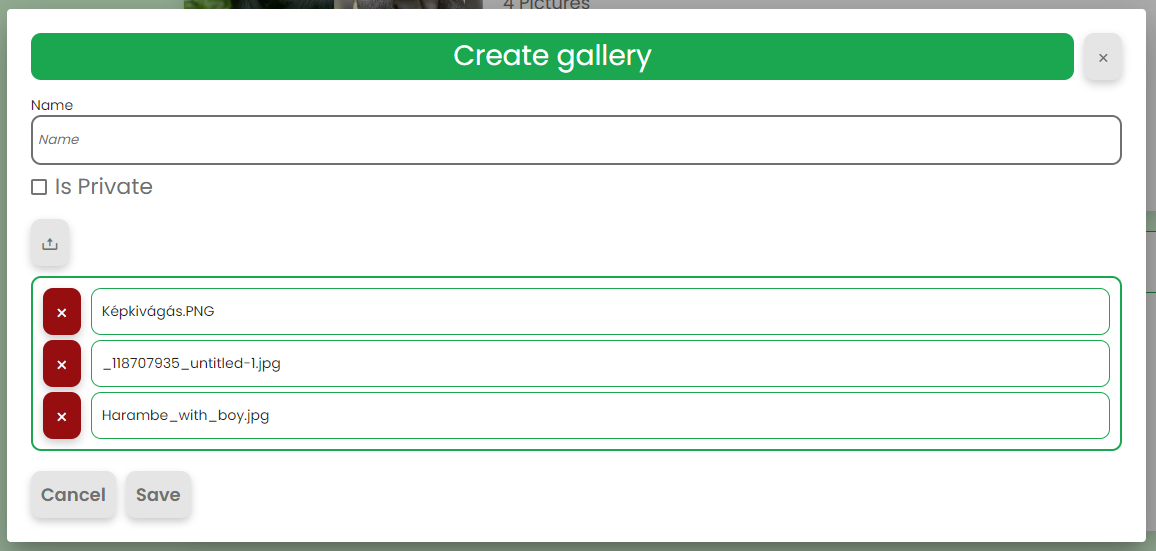
##### ConfirmComponent



16. Figure Confirm komponens

Feladata, hogy megállítsa a felhasználót egy esetleges káros művelet előtt, hogy biztosan végre akarja-e hajtani a műveletet. A pipára kattintva továbbítja a komponensnek, hogy a felhasználó beleegyezett a műveletbe.

##### AlbumDialog



17. Figure Album dialog komponens

Galériák létrehozására szolgál. Itt lehet megadni a kezdetleges adatait egy galériának, mint a neve és nyilvánosságának státusza.

Emellett megadhatóak kezdeti képek is a galériába, hogy ne utólag kelljen. Ha véletlen rosszat adtunk a listához, ki tudjuk törölni a listáról az X-re kattintva.

Ha készen vagyunk, akkor a Save-re kattintva tudjuk elmenteni a létrehozandó galériát, amit visszaküld a komponensnek, hogy indítsa el a létrehozást.

# Telepítési leírás

* Backend:  
  Visual studio 2022 telepítése ASP.NET és SQL adatbázis támogatással.  
  Az alkalmazásban Gallery.API projekt elindításával indul el a backend szerver.
* Frontend:  
  npm telepítése.  
  Segítségével a kliens könyvtárában a hivatkozott könyvtárak telepítése „npm install” paranccsal.  
  A telepítés után „npm run start” parancs elindítja a kliens alkalmazást.

# A program készítése során felhasznált eszközök

* Visual Studio 2022:
  + Felhasználás: Backend fejlesztése
* EntityFramework
  + Felhasználás: Backend adatbáziskezelési műveletek
* MSSQL
  + Felhasználás: Backend adatbázisa
* Swagger
  + Felhasználása: OpenAPI végpont specifikáció
* Visual Studio Code:
  + Frontend fejlesztése
* Angular Material:
  + Felhasználás: Frontend oldali dizájn komponensek
* Boxicon:
  + Felhasználás: Webalkalmazás ikonjainak megjelenítése
* Microsoft Word:
  + Felhasználás: Dokumentáció elkészítése
* Github:
  + Felhasználás: Verziókezelés, csapatmunka támogatása

# Összefoglalás

Implementáció során megterveztük, implementáltuk és dokumentáltuk a feladatra specifikált megoldásunkat. A létrejött alkalmazással többfelhasználós rendszerben tudjuk megtekinteni az eltárolt képeket, és tudunk velük interakciókat végrehajtani.

Az implementáció kliens-szerver architektúrát követi.

Egy dotnetes háromrétegű architektúrára épülő szerverrel, ami tartalmaz adatelérési réteget, üzleti logikai réteget és egy API réteget. Az adatelérési réteg egy MSSQL adatbázis segítségével tárolja az alkalmazásban használt adatokat.

Kliensoldali alkalmazása egy Angular alapú webalkalmazás, ami a rá jellemző MVC-re hasonlító architektúra segítségével megjeleníti az alkalmazás felületét. Mivel web alapú a kliensoldali alkalmazás, ezért platformfüggetlenül futtatható.

Az implementációs munkánk során megterveztük és megbeszéltük a funkciók részeit és részleteit, amit párhuzamosan implementáltunk backend és frontend oldalon is. Végeredményként egy saját stílusra épülő funkciókban teljes webalkalmazást készítettünk el. Az alkalmazás megbízhatóan teljesíti a követelményeket, így megfelel a felhasználó alapvető igényeinek.

# Továbbfejlesztési lehetőségek

Az elkészített alkalmazás bár funkcióiban teljes, mégsem adja meg a teljes galériaböngészési élményt. Számos funkcióval lehetne még bővíteni.

A felhasználói interakciók korlátozottak, nem adják meg a teljes közösségi élményt. Ezáltal lehetne fokozni az alkalmazást többféle közösségi funkcióval, például hozzászólási lehetőség biztosítása galériákra, esetleg képekre és felhasználókra is. Emellett a felhasználók közötti beszélgetési lehetőség, legyen privát vagy publikus, hasonlóan tudná fokozni a felhasználók elégedettségét.

A galériák nem feltétlen lehetnének statikus képek, esetleg mozgóképek vagy videók feltöltésére és megtekintésére is kialakítható lenne a felület.

Egyszerű bővítési opció lenne egy értesítési funkció bevezetése is, ami jelzi a felhasználónak, ha kedvenc galériáihoz új kép lett hozzáadva.

Lehetne bővíteni a szerver által kezelt adatok mennyiségét és minőségét is, például dátumok kezelésével. Ezek a kiegészítő adatok segítségével a felhasználóknak esetleg statisztikák is kiadhatóak lennének.

# Hivatkozások

[1. Figure A szerver architektúrája 3](#_Toc120441258)

[2. Figure Az alkalmazás adatbázis sémája 5](#_Toc120441259)

[3. Figure Login komponens 12](#_Toc120441260)

[4. Figure Register komponens 13](#_Toc120441261)

[5. Figure Loading komponens 14](#_Toc120441262)

[6. Figure Sidebar komponens 15](#_Toc120441263)

[7. Figure Profile komponens 16](#_Toc120441264)

[8. Figure My galleries komponens 17](#_Toc120441265)

[9. Figure Gallery komponens 18](#_Toc120441266)

[10. Figure Preview komponens 19](#_Toc120441267)

[11. Figure Browse komponens 20](#_Toc120441268)

[12. Figure User Gallery komponens 21](#_Toc120441269)

[13. Figure User Browse navigációk 21](#_Toc120441270)

[14. Figure User browse részletek navigációi 21](#_Toc120441271)

[15. Figure Favorites komponens 22](#_Toc120441272)

[16. Figure Confirm komponens 22](#_Toc120441273)

[17. Figure Album dialog komponens 23](#_Toc120441274)