ZÁRÓDOLGOZAT

Készítették:

Szabó Gergely László - Huszti Attila

Konzulens:

Farkas Zoltán

Miskolc

2025.

Miskolci SZC Kandó Kálmán Informatikai Technikum

Miskolci Szakképzési Centrum

SZOFTVERFEJLESZTŐ- ÉS TESZTELŐ SZAK

Szakdolgozat

Szabó Huszti Duó

Szabó Gergely László - Huszti Attila

2024-2025

Tartalomjegyzék

Felhasznált technológiák

A projektünk készítésekor többféle programot és fejlesztői környezetet is felhasználtunk.

MySQL - Adatbázis

A MySQL egy nyílt forráskódú, többfelhasználós, többszálú, SQL-alapú relációs adatbázis-kezelő szerver, melyet adatok tárolására és lekérdezésre használnak. SQL nyelv segítségével lehet adatbázist létrehozni, adatokat lekérni, és módosításokat végrehajtani a táblákon. A MySQL egy népszerű RDBMS webes alkalmazásokhoz, mert megbízható és gyors.

Entity Framework Core – Backend

Az EFC egy modern objektum-adatbázis leképező a .NET rendszerhez. Leegyszerűsíti a különböző adatbázisok használatát erősen gépelt .NET objektumokkal és a LINQ támogatásával. Az Entity Framework a legtöbb adatbázisrendszert támogatja, pl.: MySQL, SQLite, PostgreSQL, MariaDB.

C# - Backend

A C# egy objektumorientált programozási nyelv. Az objektumorientált programozás négy alapelve:

Absztrakció: Az entitások releváns attribútumainak és interakcióinak modellezése osztályokként a rendszer absztrakciós ábrázolásának meghatározásához.

Beágyazás: Elrejti az objektumok belső állapotát és funkcióit, és csak nyilvános függvénykészleten keresztül engedélyezi a hozzáférést.

Öröklési képesség: új absztrakciók létrehozására a meglévő absztrakciók alapján.

Polimorfizmus: Az öröklött tulajdonságok vagy metódusok implementálása különböző módokon több absztrakcióban.

Asp.Net Core API – Backend

Az ASP.NET egy nyílt forráskódú szerveroldali webalkalmazás-keretrendszer, amelyet dinamikus weboldalak előállítására fejlesztettek. http protokollt használ a kliens-weboldali kommunikációhoz, az adatok hozzáférése érdekében.

HTML – Frontend

A HTML nem programozási, hanem leíró nyelv, melyet weboldalak készítéséhez fejlesztettek. A Hypertext Markup Language rövidítése. Mint minden kódnyelvet, ezt is írhatjuk akár jegyzettömbben is. A hipertext jelenti az interneten található dokumentumokat (oldalakat), amelyek szöveg, két, videó, hang, animáció, vagy ezek kombinációjából épülnek fel.

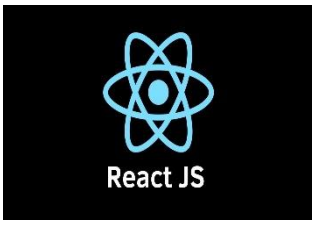
CSS - Frontend

A Cascade Style Sheets rövidítése. Az egyik legfontosabb technológia a weboldalak vizuális kialakításában. A HTML oldalak kinézetét, stílusát az itt megadott feltételekkel tudjuk befolyásolni, és lehetővé teszi különböző elemek, képek, animációk, videók elhelyezését a webes dokumentumainkba. A webböngészők megvizsgálják a HTML oldal CSS kódját, és ez alapján jelenítik meg a különböző elemeket.

Javasript - Frontend

A Javasript a webfejlesztés alapvető nyelve. Jellemzően böngészőben futtaták, frontend és backend fejlesztésre is alkalmazható. Kliensoldali szkriptnyelv, amivel interaktívvá tehetjük a weboldalainkat, ezen kívül node.js segítségével szerveroldali alkalmazások is készíthetők vele.

React – Frontend



A React a Javascript-könyvtárak egyike. Nyílt forráskóddal rendelkezik, így bárki használhatja a projektjei során. Interaktív felhasználói felületek létrehozására szolgál. Ez egy frontend-könyvtár, ami lehetővé teszi összetett felhasználói felületek létrehozását vagy felépítését kis, elszilegetelt kódrészletek, ún. komponensek felhasználásával. Az összetevők fő előnye, hogy egyetlen komponens módosítása nem érinti a teljes alkalmazást.

React Router – Frontend

React alkalmazások útvonaltervezéséhez és navigációjához használt könyvtár. Ez azt jelenti, hogy az URL cím és az alkalmazásunk összhangban van, így a megfelelő komponenst jeleníti meg. Használható egyetlen komponens megjelenítésére, vagy bizonyos részeket (pl. Navbar, Footer, stb.) statikusak maradnak, ami azt jelenti, hogy a felhasználó számára minden aloldalunkról látható és elérhető marad. Ennek segítségével csak az oldal tartalmi része rendelelődik újra, aminek köszönhetően gyorsabban tölt be az oldalunk.

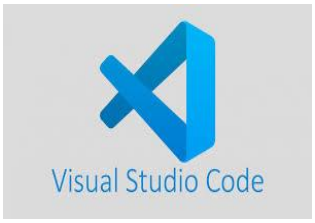
Fejlesztői környezetek

XAMPP

A XAMPP egy olyan szoftvercsomag, ami többek között Apache webszervert és MySQL adatbáziskezelőt, PHP-t tartalmaz, így könnyen telepíthető és üzembe helyezhető az adatbázis-fejlesztői környezet. A XAMPP segítségével a fejlesztők lokálisan tudnak webalkalmazásokat fejleszteni és tesztelni.

Használatával könnyedén létre tudunk hozni egy teljes webkiszolgáló környezetet a saját számítógépünkön, és tesztelhetjük a weboldalunk működését, illetve könnyedén kezelhetjük a kapcsolódó adatbázisunkat.

Visual Studio Code



A Visual Studio Code (rövid nevén VS Code) egy nyílt forráskódú, integrált fejlesztői környezet, melyet A Microsoft fejlesztett ki Windows, Linux, macOS és webböngészők számára. A funkciók közé tartozik a hibakeresés támogatása, a szintaxis kiemelése, az intelligens kódkészítés, a kódrészletek, és a kód újrafeldolgozása, valamint a Git beágyazott verziókezelése. A VS Code alkalmas különböző programozási nyelvekhez, keretrendszerekhez, és a fejlesztők egyedi bővítményekkel is testre szabhatják a munkakörnyezetüket.

Visual Studio 2022

A Visual Studio 2022 egy többplatformos fejlesztői környezet, amit elsősorban Windows operációs rendszerre terveztek. Segítségével hatékonyan készíthetünk alkalmazásokat, beleértve az asztali alkalmazásokat, webalkalmazásokat és szolgáltatásokat. A VS 2022 több programozási nyelvet támogat, beleértve a projektünkhöz használt C# nyelvet. Ebben a fejlesztői környezetben lehetőségünk van kódírásra, hibakeresésre, tesztelésre, továbbá vizuális tervezésre.

Swagger

A Swagger egy nyílt forráskódú keretrendszer az API leírására, egy mindenki számára érthető, közös nyelven. RESTful APIk tervezésére, építésére és dokumentációjára használjuk. Érdemes úgy tekinteni rá, mint egy ház tervrajzára. A dokumentum JSON vagy YAML formátumban készül, bemutatja az API végpontjait és az általuk elfogadott paramétereket, valamint az API által elfogadott válasz- és hibakódokat. Köszönhetően annak, hogy több programozási nyelvet és keretrendszert is támogat, lehetővé teszi a programozók számára, hogy csökkentsék a hibák számát, így elérve a hatékonyabb munkavégzést a fejlesztés során.

Kommunikációs felületek

A projekt során többféle kommunikációs felületet használtunk attól függően, éppen melyik státuszban jártunk a felépítés során. A lentiekben ezeket fogjuk bemutatni.

Trello

A Trello egy webalapú, Kanban stílusú listakészítő alkalmazás.Itt táblázatos formában tudtuk meghatározni az éppen aktuális feladatokat, határidőket megjelölni, illetve az elkészült részfeladatokat megosztani a csapatban.

GitHub

A GitHub Inc. egy amerikai nemzetközi vállalat, amely a Git segítségével szoftverfejlesztési és verziókövetés szolgáltatást nyújt. A projektünk nagyobb részelemeit ide publikáltuk, így a teljes csapat számára nyomon követhető volt, ki hol jár a vállalt feladatával.

🎬 Projekt adatbázis

Ez az adatbázis egy filmes tematikájú rendszerhez készült, amely színészeket, rendezőket, filmeket és a felhasználók értékeléseit kezeli. Az alábbiakban részletesen bemutatjuk a táblákat, azok szerepét és kapcsolatait.

💼 1. actor – Színészek táblája

Ez a tábla tartalmazza az összes nyilvántartott színész adatait.

Oszlop neve	Típus	Leírás
Id	char(36)	Egyedi azonosító (UUID)
Name	varchar(40)	Színész neve
Nationality	varchar(40)	Nemzetiség
Birthday	date	Születési dátum
Oscar_award	tinyint(1)	Oscar-díj (1 = igen, 0 = nem)
Sex	enum	Neme: Male vagy Female
ProfilKép	varchar(256)	Profilkép URL vagy fájlnév

☞ **Célja**: a színészek életrajzi adatainak tárolása és megjelenítése.

🎬 2. director – Rendezők táblája

Ez a tábla hasonló felépítésű, mint a színészeké, de rendezőkre vonatkozik.

Oszlop neve	Típus Leírás
Id, Name, Nationality, Birthday, Oscar_award, Sex, ProfilKép	

☞ **Célja**: a filmek rendezőinek adatainak nyilvántartása.

📽️ 3. films – Filmek táblája

A filmek minden alapadatát tárolja.

Oszlop neve	Típus	Leírás
Id	char(36)	Egyedi filmazonosító
Name	varchar(40)	Film címe

Oszlop neve	Típus	Leírás
Director	char(36)	Rendező azonosítója (director.Id)
Genre	varchar(16)	Műfaj (pl. Drama, Action)
ReleaseYear	int(8)	Megjelenés éve (hibás formátum is lehet)
Length	int(4)	Hossz percben
AgeCertificates	int(2)	Korhatár besorolás (pl. 12, 16, 18)
Summary	text	Rövid tartalom
Kép	varchar(256)	Poszter vagy kép elérési útja

☞ **Célja**: a filmek metaadatainak tárolása.

🎭 4. filmactor – Film-színész kapcsolat (sok-sok kapcsolat)

Ez egy **kapcsolótábla**, amely megadja, hogy melyik színész szerepel melyik filmben.

Oszlop neve	Típus	Leírás
FilmId	char(36)	Film azonosító (films.Id)
ActorId	char(36)	Színész azonosító (actor.Id)

☞ **Kapcsolat**: Egy filmhez több színész tartozhat, és egy színész több filmben is szerepelhet.

★ 5. rating – Értékelések

Ez a tábla a felhasználók által adott filmértékeléseket tartalmazza.

Oszlop neve	Típus	Leírás
Id	char(36)	Értékelés egyedi azonosító
FilmId	char(36)	A film, amit értékelnek
UserId	char(36)	Értékelő felhasználó azonosítója
Review	int(11)	Értékelés (pl. 1–10 skálán)

☞ **Célja**: a filmek minősítésének rögzítése felhasználói szinten.

👥 6. user – Felhasználók

Az alkalmazás felhasználóinak adatait tárolja.

Oszlop neve	Típus	Leírás
Id	char(36)	Felhasználó azonosító
Name	varchar(40)	Teljes név
FelhasznaloNev	varchar(100)	Bejelentkezési név
Hash	varchar(64)	Jelszó hash
Email	varchar(40)	Email-cím
Sex	enum	Neme
Joined	date	Regisztráció dátuma
Role	int(10)	Szerepkör (role.Id)
ProfilKép	varchar(256)	Profilkép URL vagy fájlnév

☞ **Megjegyzés**: a jelszavak nem sima szövegként, hanem hash formátumban vannak tárolva a biztonság érdekében.

🛡 7. role – Felhasználói szerepek

Ez a tábla különböző jogosultsági szinteket definiál.

Oszlop neve	Típus	Leírás
Id	int(10)	Szerepkör azonosító
Name	varchar(10)	Szerepkör neve (pl. Admin, User, Guest)
Description	varchar(40)	Leírás

☞ **Kapcsolat**: a user tábla Role mezője erre a táblára hivatkozik.

🔗 Kapcsolatok összefoglalása

- Egy filmnek **egy rendezője** van (films.Director → director.Id).
- Egy filmben **több színész** is játszhat (filmactor kapcsolótábla).
- Egy felhasználó **több filmet is értékelhet** (rating).
  - Egy felhasználónak van egy **szerepköre**, amely meghatározza a jogosultságát (user.Role → role.Id).

Back-end Dokumentáció

1. Általános Controller Működése

Az alkalmazás több CRUD típusú controllert tartalmaz (ActorController, DirectorController, FilmController, UserController,RatingController). Ezek azonos szerkezetben működnek és az alábbi műveleteket valósítják meg:

- GET /[controller]: Az összes objektum lekérdezése

- GET /[controller]/{id}: Egy adott objektum lekérdezése ID alapján

- POST /[controller]: Új objektum létrehozása

- PUT /[controller]/{id}: Objektum frissítése

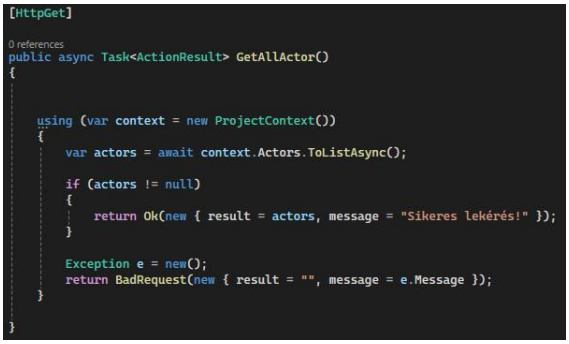
- DELETE /[controller]/{id}: Objektum törlése

A POST és PUT metódusok az Actor- , Director- , User- és FilmControllerben képfeltöltést is kezelnek, amely FTP-n keresztül történik. A képek URL-je bekerül az ojektumok 'ProfilKép' vagy ‘Kép’mezőjébe.

2. Példa: ActorController működése

2.1 GET /actor

Ez a végpont visszaadja az összes színész objektumot az adatbázisból. A válasz egy JSON tömb, amely tartalmazza a színészek minden adatát.



```
[HttpGet]

0 references
public async Task<ActionResult> GetAllActor()
{


    using (var context = new ProjectContext())
    {
        var actors = await context.Actors.ToListAsync();

        if (actors != null)
        {
            return Ok(new { result = actors, message = "Sikeres lekérés!" });
        }

        Exception e = new();
        return BadRequest(new { result = "", message = e.Message });
    }
}
```

2.2 GET /actor/{id}

Ez a végpont egy adott színészt ad vissza az ID alapján. Ha a megadott ID nem létezik, akkor NotFound hibakóddal tér vissza (404). Az adott színész minden adatát visszadja.

```
[HttpGet("id")]

0 references
public async Task<ActionResult> GetActorId(Guid id)
{
    using (var context = new ProjectContext())
    {
        var actor = await context.Actors.FirstOrDefaultAsync(x => x.Id == id);

        if (actor != null)
        {
            return Ok(new { result = actor, message = "Sikeres lekérés!" });
        }
        return NotFound(new { result = "", message = "Nincs ilyen Actor az adatbázisban!" });
    }
}
```

2.3 POST /actor

Ez a végpont új színész létrehozását teszi lehetővé. A kérés tartalmaz egy multipart/form-data típusú body-t, amely a színész adatait tartalmaza. A kép feltöltése FTP-n keresztül történik, és a fájl URL-je kerül mentésre az adatbázisban.

```
[Authorize(Roles = "1")]
[HttpPost]

0 references
public async Task<ActionResult> AddNewActor([FromForm] CreateActorDTO createActorDTO)
{

    string fileName = "users.png";
    string subFolder = "";

    var url = "ftp://ftp.nethely.hu" + "/" + subFolder + "/" + fileName;

    if (createActorDTO.Kep != null)
    {
        fileName = Path.GetFileName(createActorDTO.Kep.FileName);
        url = "ftp://ftp.nethely.hu" + "/" + subFolder + "/" + fileName;

        FtpWebRequest request = (FtpWebRequest)WebRequest.Create(url);
        request.Credentials = new NetworkCredential("backend_kalahora", "!SZGL09021992");
        request.Method = WebRequestMethods.Ftp.UploadFile;
        await using (Stream ftpStream = request.GetRequestStream())
        {
            await createActorDTO.Kep.CopyToAsync(ftpStream);
        }

    }


    var actor = new Actor
    {
        Id = Guid.NewGuid(),
        Name = createActorDTO.Name,
        Nationality = createActorDTO.Nationality,
        Birthday = createActorDTO.Birthday,
        OscarAward = createActorDTO.OscarAward,
        Sex = createActorDTO.Sex,
        ProfilKép = $"http://kepek.noreplykalahora1992.nhely.hu/{fileName}"
    };

    using (var context = new ProjectContext())
    {
        await context.Actors.AddAsync(actor);
        await context.SaveChangesAsync();

        return StatusCode(201, new { result = actor, message = "Sikeres felvétel!" });
    }
}
```

Egy CreateActorDTO alapján hozza létre az új Actor-t, melynek Profilképe ha nincs megadva egy alap ‘user.png’ lesz. Csak Role 1-es Token-nel lehet meghívni, amely az Admin felhasználó.

```
[Authorize(Roles = "1")]
[HttpDelete]

0 references
public async Task<ActionResult> DeleteActor(Guid id)
{
    using (var context = new ProjectContext())
    {
        var actor = await context.Actors.FirstOrDefaultAsync(x => x.Id == id);

        if (actor != null)
        {
            context.Actors.Remove(actor);
            await context.SaveChangesAsync();

            return Ok(new { result = actor, message = "Sikeres törlés!" });
        }

        return NotFound(new { result = "", message = "Nincs ilyen Actor az adatbázisban!" });
    }
}
```

2.4 PUT /actor/{id}

Ez a végpont meglévő színész adatainak frissítésére szolgál. A body hasonló a POST metóduséhoz, szintén képfeltöltést is kezel. Csak akkor történik frissítés, ha az ID létezik, és ez alapján találja meg az adott színészt, ha nincs akkor NotFound hibaüzenetet ad vissza.

```
[Authorize(Roles = "1")]
[HttpPut]

0 references
public async Task<ActionResult> UpdateActor(Guid id, [FromForm] UpdateActorDTO updateActorDTO)
{
    using (var context = new ProjectContext())
    {

        string fileName = "users.png";

        if (updateActorDTO.Kep != null)
        {


            string subFolder = "";
            fileName = Path.GetFileName(updateActorDTO.Kep.FileName);
            var url = "ftp://ftp.nethely.hu" + "/" + subFolder + "/" + fileName;

            FtpWebRequest request = (FtpWebRequest)WebRequest.Create(url);
            request.Credentials = new NetworkCredential("backend_kalahora", "!SZGL09021992");
            request.Method = WebRequestMethods.Ftp.UploadFile;
            await using (Stream ftpStream = request.GetRequestStream())
            {
                await updateActorDTO.Kep.CopyToAsync(ftpStream);
            }

        }


        var existingActor = await context.Actors.FirstOrDefaultAsync(x => x.Id == id);

        if (existingActor != null)
        {
            existingActor.Name = updateActorDTO.Name;
            existingActor.Nationality = updateActorDTO.Nationality;
            existingActor.OscarAward = updateActorDTO.OscarAward;
            existingActor.Birthday = updateActorDTO.Birthday;
            existingActor.Sex = updateActorDTO.Sex;
            existingActor.ProfilKép = $"http://kepek.noreplykalahora1992.nhely.hu/{fileName}";

            context.Actors.Update(existingActor);
            await context.SaveChangesAsync();

            return Ok(new { result = existingActor, message = "Módosítás sikeres!" });
        };

        return NotFound(new { result = "", message = "Nincs ilyen Actor az adatbázisban!" });
    }
}
```

2.5 DELETE /actor/{id}

Ez a végpont egy adott színész törlését végzi el ID alapján. Egy GUID tipusú ID-t kap, majd az megkeresve törli az adatbázisból, ha nem találja egy NotFound hibakóddal tér vissza (404). Csak Role 1-es Token-nel lehet meghívni, amely az Admin felhasználó.

3. Egyéb: Controller működése

3.1 Get /Rating AvrgRating

Ez a végpont a filmekhez tartozó értékelések átlagát számítja ki és adja vissza. Az értékelések alapján számított átlagos értékelést a filmhez tartozó összes értékelés (Ratings) alapján kalkulálja. Ha a filmhez nincs értékelés, akkor az átlagos értékelés 0 lesz. A végpont a filmId alapján lekérdezi a filmet az adatbázisból, majd ha a film létezik, az összes hozzá tartozó értékelést összegyűjti és kiszámítja azok átlagát.

Ha a film nem található, akkor a válasz egy hibát tartalmaz, amely tájékoztatja a felhasználót, hogy a megadott film nem létezik az adatbázisban.

```
[HttpGet("Rating")]

0 references
public async Task<ActionResult> AvrgRating(Guid id)
{
    using (var context = new ProjectContext())
    {
        var film = await context.Films
            .Include(x => x.Ratings)
            .FirstOrDefaultAsync(x => x.Id == id);

        if (film == null)
        {
            return BadRequest(new { result = "", message = "Nincs ilyem film az adatbázisban!" });
        }

        var atlagRating = film.Ratings.Any() ? film.Ratings.Average(r => r.Review) : 0;

        return Ok(new { result = atlagRating});

    }
}
```

3.2 POST /FtpServer

Ez az API végpont lehetővé teszi fájlok feltöltését egy FTP szerverre. A fájlt a kérésben (multipart form-data formátumban) kell elküldeni, és az API feltölti azt a megadott FTP szerverre. A fájl neve és elérhetősége a válaszban kerül visszaküldésre.

3.3 POST /Login

A bejelentkezési kéréshez **FelhasznaloNev** és **Hash** adatokat kell küldeni. **FelhasznaloNev** a felhasználó neve, amelyet a regisztrálásakor használtak. **Hash** a felhasználó jelszavának **SHA256**-os hash-elt formája.. A backend ellenőrzi, hogy a megadott felhasználónevet és hash-elt jelszót tartalmazó rekord létezik-e az adatbázisban. Ha igen, akkor generál egy **JWT** tokent a felhasználó számára, és visszaküldi a felhasználó adatait, valamint a generált tokent. Ha a bejelentkezési adatok nem megfelelőek (pl. hibás felhasználónév vagy jelszó), akkor hibát jelez. Az autentikációval kapcsolatos információk, mint például a **JWT token**, a

válaszban kerülnek visszaadásra, amelyet a felhasználó később felhasználhat a rendszer más végpontjainak elérésére.

```
[HttpPost]

0 references
public async Task<ActionResult> Login(LoginDTO loginDTO)
{
    if (string.IsNullOrEmpty(loginDTO?.FelhasznaloNev) || string.IsNullOrEmpty(loginDTO?.Hash))
    {
        return BadRequest("Felhasználónév és jelszó megadása kötelező.");
    }

    using (var context = new ProjectContext())
    {
        try
        {
            string Hash = Program.CreateSHA256(loginDTO.Hash);

            var loggedUser = await context.Users
                .FirstOrDefaultAsync(f => f.FelhasznaloNev == loginDTO.FelhasznaloNev && f.Hash == Hash);
            if (loggedUser != null)
            {
                var jwtToken = _tokenGenerator.GenerateToken(loggedUser);

                lock (Program.LoggedInUsers)
                {
                    Program.LoggedInUsers.Add(jwtToken, loggedUser);
                }

                return Ok(new LoggedUser { Id = loggedUser.Id, Name = loggedUser.Name, Email = loggedUser.Email, Jog = loggedUser.Role, Token = jwtToken });
            }

            else
            {
                return BadRequest("Hibás név vagy jelszó!");
            }
        }
        catch (Exception ex)
        {
            return BadRequest(ex.Message);
        }

    }
}
```

3.4 POST /LogOut

A felhasználó kijelentkezését a POST /Logout/{uId} végpont végzi. A kérés paramétereként a felhasználó egyedi azonosítóját (uId) kell megadni. Az alkalmazás megvizsgálja, hogy a megadott uId szerepel-e a Program.LoggedInUsers szótárban (ami a bejelentkezett felhasználók adatainak tárolására szolgál).

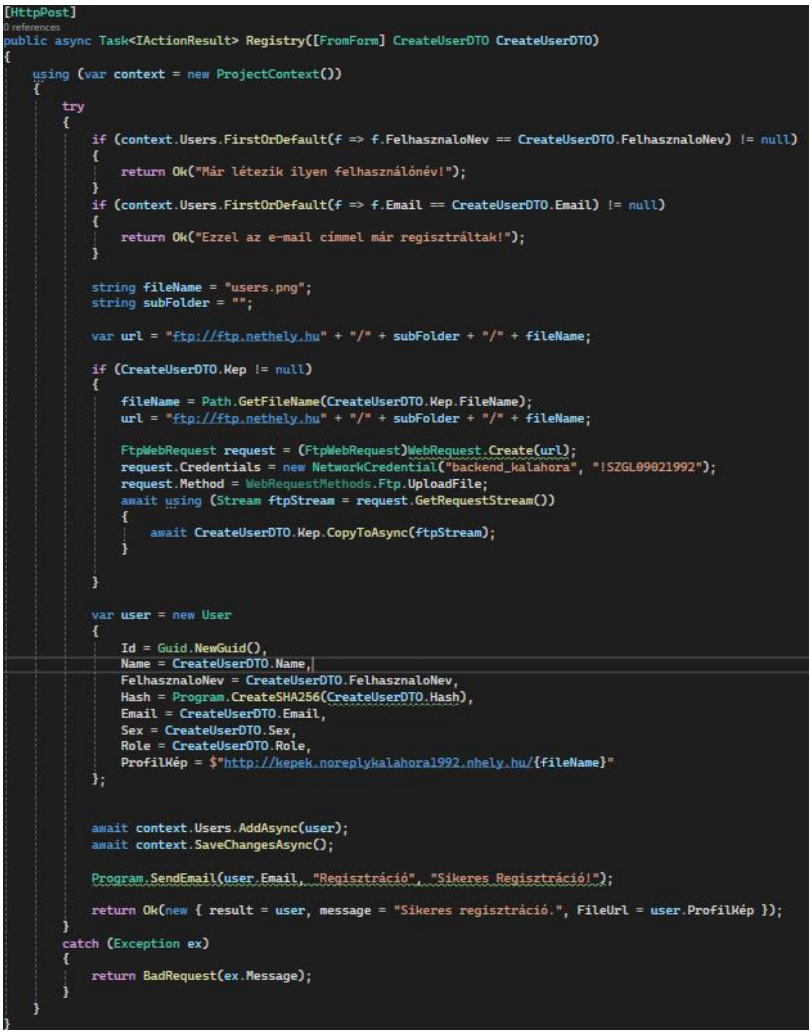
```
[ApiController]
0 references
public class LogOutController : ControllerBase
{
    [HttpPost("{uId}")]
    0 references
    public IActionResult Logout(string uId)
    {


        if (Program.LoggedInUsers.ContainsKey(uId))
        {
            Program.LoggedInUsers.Remove(uId);
            return Ok("Sikeres kijelentkezés");
        }
        else
        {
            return BadRequest("Sikertelen kijelentkezés!");
        }
    }
}
```

3.5 POST /Registry

Ez az API végpont új felhasználó regisztrációját végzi el. A felhasználónak meg kell adnia a szükséges adatokat, mint például felhasználónév, e-mail cím, jelszó, és opcionálisan egy profilkép. A regisztrációs adatokat az adatbázisban tárolja, és a felhasználót értesíti a sikeres regisztrációról egy e-mailben. Az API először ellenőrzi, hogy a megadott felhasználónév és e-mail cím már létezik-e az adatbázisban. Ha bármelyik létezik, hibát jelez és nem folytatja a regisztrációt. a nincs hiba, a profilkép feltöltése történik, amennyiben a felhasználó biztosít képet. A kép FTP szerverre kerül feltöltésre, és az URL a válaszban szerepel. A felhasználó

adatait egy új User objektumban tárolja az adatbázisban, majd sikeres regisztráció esetén visszaküld egy pozitív választ a felhasználó adataival és a profilkép URL-jével. Az alkalmazás egy e-mailt küld a regisztrált e-mail címre a sikeres regisztrációról.



```
[HttpPost]
0 references
public async Task<IActionResult> Registry([FromForm] CreateUserDTO CreateUserDTO)
{
    using (var context = new ProjectContext())
    {
        try
        {
            if (context.Users.FirstOrDefault(f => f.FelhasznaloNev == CreateUserDTO.FelhasznaloNev) != null)
            {
                return Ok("Már létezik ilyen felhasználónév!");
            }
            if (context.Users.FirstOrDefault(f => f.Email == CreateUserDTO.Email) != null)
            {
                return Ok("Ezzel az e-mail címmel már regisztráltak!");
            }

            string fileName = "users.png";
            string subFolder = "";

            var url = "ftp://ftp.nethely.hu" + "/" + subFolder + "/" + fileName;

            if (CreateUserDTO.Kep != null)
            {
                fileName = Path.GetFileName(CreateUserDTO.Kep.FileName);
                url = "ftp://ftp.nethely.hu" + "/" + subFolder + "/" + fileName;

                FtpWebRequest request = (FtpWebRequest)WebRequest.Create(url);
                request.Credentials = new NetworkCredential("backend_kalahora", "!SZGL09021992");
                request.Method = WebRequestMethods.Ftp.UploadFile;
                await using (Stream ftpStream = request.GetRequestStream())
                {
                    await CreateUserDTO.Kep.CopyToAsync(ftpStream);
                }

            }

            var user = new User
            {
                Id = Guid.NewGuid(),
                Name = CreateUserDTO.Name,
                FelhasznaloNev = CreateUserDTO.FelhasznaloNev,
                Hash = Program.CreateSHA256(CreateUserDTO.Hash),
                Email = CreateUserDTO.Email,
                Sex = CreateUserDTO.Sex,
                Role = CreateUserDTO.Role,
                ProfilKép = $"http://kepek.noreplykalahora1992.nhely.hu/{fileName}"
            };


            await context.Users.AddAsync(user);
            await context.SaveChangesAsync();

            Program.SendEmail(user.Email, "Regisztráció", "Sikeres Regisztráció!");

            return Ok(new { result = user, message = "Sikeres regisztráció.", FileUrl = user.ProfilKép });
        }
        catch (Exception ex)
        {
            return BadRequest(ex.Message);
        }
    }
}
```

4. DTO:

4.1 DTO(s)

A kód **DTO-kat** definiál, amelyek különböző objektumok, mint például színész, rendező, film, felhasználó és értékelés kezelésére szolgálnak az alkalmazásban. Ezek a DTO-k segítenek az adatok kezelésében, amelyeket az API végpontokon keresztül küldünk és fogadunk.

```
//Actor DTO(s)
1 reference
public record CreateActorDTO(string Name, string Nationality, DateTime Birthday, bool OscarAward, string Sex, IFormFile? Kep);
1 reference
public record UpdateActorDTO(string Name, string Nationality, DateTime Birthday, bool OscarAward, string Sex, IFormFile? Kep);

//Director DTO(s)
1 reference
public record CreateDirectorDTO(string Name, string Nationality, DateTime Birthday, bool OscarAward, string Sex, IFormFile? Kep);
1 reference
public record UpdateDirectorDTO(string Name, string Nationality, DateTime Birthday, bool OscarAward, string Sex, IFormFile? Kep);

//Film DTO(s)
1 reference
public record CreateFilmDTO(string Name, Guid Director,string Genre, int ReleaseYear, int Length, int? Reviews, int AgeCertificates, string Summary, IFormFile? Kep);
1 reference
public record UpdateFilmDTO(string Name, Guid Director, string Genre, int ReleaseYear, int Length, int? Reviews, int AgeCertificates, string Summary, IFormFile? Kep);

//User DTO(s)
2 references
public record CreateUserDTO(string Name,string FelhasznaloNev, string? Hash, string Email, string Sex, DateTime? Joined, int Role, IFormFile? Kep);
1 reference
public record UpdateUserDTO(string Name, string FelhasznaloNev, string Hash, string Email, string Sex, DateTime? Joined, int Role, IFormFile? Kep);

//Rating DTO(s)
1 reference
public record RatingDTO(Guid FilmId, Guid UserId, int Review);

//Login DTO(s)
1 reference
public record LoginDTO(string FelhasznaloNev, string Hash);
```

4.2 LoggedUser

A LoggedUser osztály egy adatátviteli objektum (DTO), amely a sikeres bejelentkezés után tartalmazza a felhasználó adatait, beleértve az azonosítót, nevet, email címet, szerepkört és a generált JWT (JSON Web Token) tokent. Ez az osztály segít a felhasználói adatkezelésben, különösen a bejelentkezési munkamenetek kezelésében, és a kliens számára biztosítja az autentikációs tokent a további API hívásokhoz.

```
public class LoggedUser
{
    1 reference
    public Guid Id { get; set; }
    1 reference
    public string? Name { get; set; }
    1 reference
    public string? Email { get; set; }
    1 reference
    public int Jog {  get; set; }
    1 reference
    public string? Token {  get; set; }
}
```

5. Models:

5.1 Actor

Az Actor osztály a rendszerben tárolt színészek adatait reprezentálja. Az osztály tartalmazza a színész személyes adatait (név, születési dátum, nem, oscar díj) és a színészhez kapcsolódó filmeket.

5.2 Director

A Director osztály a filmrendezők adatainak tárolására szolgál. Minden egyes rendező rendelkezik alapvető adatokkal, mint például név, nemzetiség, születési dátum, oscar-díj, nem és profilkép. Ezen kívül a rendezőkhez tartozhatnak filmek, amelyekkel kapcsolatban rendeztek.

5.3 Film

A Film osztály a filmek adatainak tárolására szolgál az alkalmazásban. Minden egyes film egyedi jellemzőkkel rendelkezik, mint például a cím, rendező, műfaj, megjelenési év, hosszúság, életkori besorolás és egyéb információk. Ezen kívül egy-egy filmhez tartozhatnak értékelések ,rendezők és szereplők, amelyeket a Film osztály kapcsolati tulajdonságai kezelnek..

5.4 JwtOptions

A JwtOptions osztály a JSON Web Token (JWT) konfigurációs beállításait tárolja. Ez az osztály három fő paramétert tartalmaz, amelyek segítenek a JWT érvényesítésében és generálásában. Az osztály célja, hogy közvetítse a szükséges információkat az alkalmazásban történő JWT kezeléséhez, mint például az érvényesítő és célzott közönség meghatározása, valamint a titkos kulcs, amely a token aláírását szolgálja.

5.5 ProjectContext

A ProjectContext osztály az Entity Framework Core DbContext osztályból öröklődik, és a projekt adatbázisának kezelése céljából lett létrehozva. A ProjectContext felelős az alkalmazás adatbázis kapcsolatainak és entitásainak kezeléséért, beleértve a filmek, színészek, rendezők, értékelések és felhasználók adatait. Az osztály az adatbázis lekérdezéseket és műveleteket végez, és biztosítja az adatok integritását.

5.6 Rating

A Rating osztály a felhasználók által adott értékeléseket képviseli, melyek egy adott filmhez tartoznak. Az osztály tárolja a felhasználó értékelését egy filmre vonatkozóan, amely egy numerikus érték. Ez az osztály segít az alkalmazásban a filmek értékelésének kezelésében és tárolásában.

5.7 Role

A Role osztály a felhasználói szerepköröket képviseli az alkalmazásban. Minden felhasználóhoz tartozik egy szerepkör, amely meghatározza, hogy milyen jogosultságokkal rendelkezik az adott felhasználó az alkalmazásban. A szerepkörök fontosak a hozzáférés-vezérlés szempontjából, mivel ezek biztosítják, hogy a felhasználók csak azokat az adatokat és funkciókat érhessék el, amelyekhez jogosultságuk van.

5.8 User

A User osztály egy felhasználó adatainak tárolására szolgál az alkalmazásban. Az osztály tartalmazza a felhasználói fiókhoz szükséges alapvető információkat, például a felhasználó nevét, email címét, szerepét, valamint a csatlakozás időpontját. A User osztály ezen kívül tartalmazza a felhasználó által adott értékeléseket (ratings), valamint a felhasználóhoz tartozó szerepkört (Role).

6. Services:

6.1 ITokenGenerator

Az ITokenGenerator interfész egy szerződés a JSON Web Token (JWT) generálására szolgáló szolgáltatás számára. A JWT (JSON Web Token) egy nyílt szabvány (RFC 7519), amely biztonságos adatátvitelt biztosít, és az alkalmazások hitelesítési és jogosultsági rendszereiben használatos. Ez a metódus felelős egy JSON Web Token (JWT) generálásáért a megadott felhasználó (User) adatai alapján. A token a felhasználó autentikációs és autorizációs adatait tartalmazza, és általában az alkalmazásban való bejelentkezés után kerül létrehozásra, hogy az azonosított felhasználót további kérésekhez hitelesítse.

6.2 TokenGenerator

Ez az osztály felelős a JWT tokenek generálásáért a felhasználói adatok (mint például felhasználónév, e-mail, felhasználói azonosító és szerepkör) felhasználásával, és egy titkos kulcs segítségével aláírja a tokent. A generált token biztosítja a felhasználó hitelesítését a rendszerben. A metódus létrehoz egy JWT tokent a felhasználó (User) adatai alapján. A token tartalmazza a felhasználó hitelesítési információit és szerepkörét, valamint azokat a kulcsokat, amelyek segítségével az alkalmazás biztosítani tudja a jogosultságokat és hitelesítést.

7. Program.cs:

7.1 Felhasználók Nyilvántartása

A LoggedInUsers egy statikus dictionary, amely a bejelentkezett felhasználók kezelésére szolgál. A felhasználók itt vannak tárolva, és az alkalmazás csak addig tartja őket a memóriában, amíg be vannak jelentkezve.

7.2 SHA-256 Hashing

A CreateSHA256 függvény a SHA-256 algoritmus segítségével hashalja a bemeneti adatokat. Ezt használhatjuk például jelszavak titkosítására. A jelszavakat először "salt"-eljük (a GenerateSalt függvénnyel), majd a hash értéket tároljuk el az adatbázisban.

7.3 Send Email

A SendEmail egy aszinkron metódus, amely egy emailt küld a SmtpClient használatával a Gmail SMTP szerverén keresztül. A Gmail SMTP szerverét (smtp.gmail.com) használja. A kód beállítja a hitelesítést (noreply.kalahora1992@gmail.com és az app-specifikus jelszó). A függvény paraméterként megadott címzettnek küldi el az emailt, amely tartalmazza a témát és a testet.

7.4 JWT Autentikáció

Az alkalmazás JWT (JSON Web Token) alapú autentikációt használ a felhasználói hitelesítéshez. A JwtOptions beállításokat a konfigurációból olvassuk be (Secret, Issuer, Audience). Az alkalmazás beállítja a JwtBearer hitelesítést, amely a HTTP kérésekhez csatolt JWT tokeneket érvényesíti. A TokenValidationParameters beállításokkal az alkalmazás

validálja a tokeneket, ellenőrzi az aláírást, az érvényes kibocsátókat (Issuer) és a címzettet (Audience).

7.5 CORS Beállítások

A CORS (Cross-Origin Resource Sharing) beállítások lehetővé teszik a böngészőből származó kérések kezelését a megadott forrásokból. A kód a http://localhost:3000-at engedélyezi, amely a React alkalmazás helyi fejlesztési szervere. A CORS beállítja, hogy bármilyen HTTP metódus és fejlécek elfogadottak legyenek a megadott forrásból érkező kéréseknél.

7.6 Swagger

Az alkalmazás Swagger-t használ az API dokumentálására és tesztelésére. A Swagger csak fejlesztési környezetben van engedélyezve (app.Environment.IsDevelopment()). A Swagger UI elérhetővé válik az API végpontok teszteléséhez és dokumentálásához.

7.7 HTTP Pipeline Konfigurálás

A app.UseHttpsRedirection() biztosítja, hogy az összes HTTP kérést automatikusan HTTPS-re irányítja át. Az alkalmazás app.UseAuthorization()-nal kezeli az engedélyezési szabályokat. A app.UseCors(MyAllowSpecificOrigins) biztosítja, hogy a CORS szabályok érvényesek legyenek.

Frontend dokumentáció

A frontend az alkalmazás felhasználói felülete, ahol a látogató vagy felhasználó interakcióba lép az oldallal. Az alábbiakban bemutatjuk, hogyan működik általánosan egy modern webes frontend (pl. React.js alapú) egy REST API-val kommunikáló alkalmazásban.

A frontend alkalmazás általában a következő fő részekből áll:

- **Komponensek (Components):** újrahasználható UI elemek, pl. DogList, OwnerForm, BreedSelector.
- **Oldalak (Pages):** különböző nézetek, pl. főoldal, kutyák listája, fajta szerkesztés.
- **Állapotkezelés (State Management):** pl. useState, useEffect vagy egy globális store, pl. Redux, Zustand.
- **Router (Útvonalkezelő):** navigáció a különböző oldalak között pl. React Router.

A frontend az adatokat **HTTP kérésekkel** (pl. fetch, axios) kéri le és küldi vissza a backend REST API-n keresztül. A leggyakoribb műveletek:

Ha az oldal igényel **bejelentkezést** vagy jogosultságkezelést:

- A frontend belépéskor kap egy **JWT tokent**, amit minden védett kérésnél elküld (Authorization: Bearer <token>).
- A backend ellenőrzi a token érvényességét.

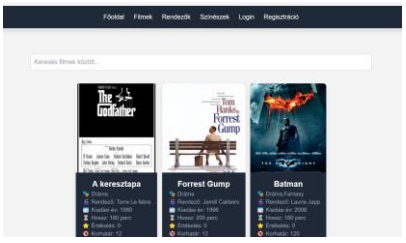
Példa a leggyakoribb műveletekhez:

Művelet	HTTP metódus	Leírás
Lekérdezés	GET	Adatok lekérése (pl. összes film listája)
Létrehozás	POST	Új adat létrehozása (pl. új film hozzáadása)
Módosítás	PUT / PATCH	Létező adat frissítése (pl. film módosítása)
Törlés	DELETE	Adat törlése (pl. egy film eltávolítása)

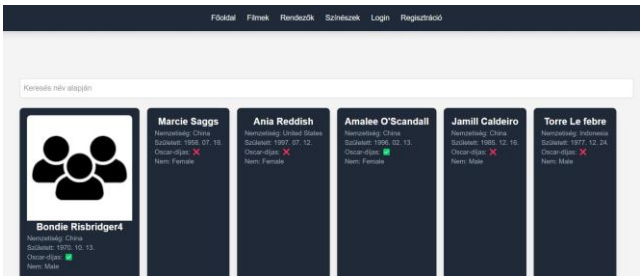
Az alábbiakban láthatjuk az webes alkalmazás oldalait, képekkel illusztrálva:

Főoldal

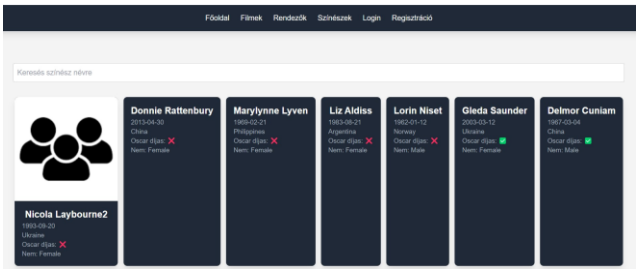
Filmek



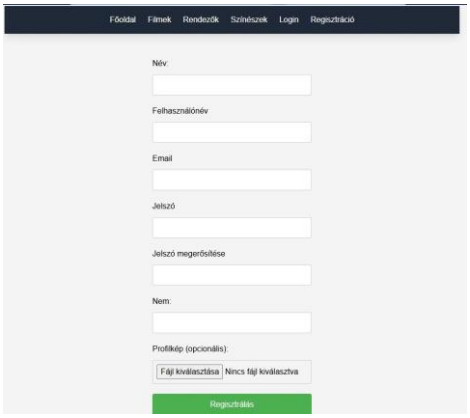
Színészek



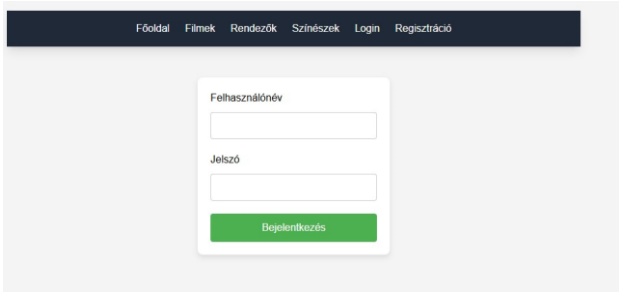
Rendezők



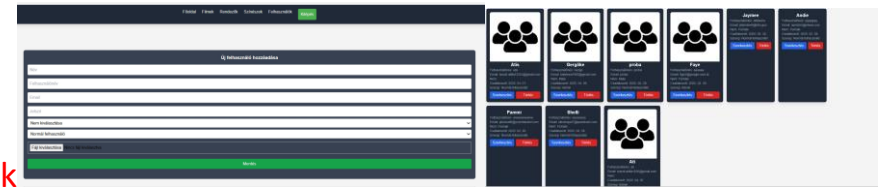
Regisztráció



Login



Felhasználók



React szerkezet

A react kód példa bemutatása az Actors page felépítésén keresztül:

A react komponens inportálása és a változók deklarálása, a react useEffect API kommunikációk által használt useState változók és a localstorageben tárolt komponensek lekérése, a localstorageban tároljuk a felhasználók Id-ját és jogkörét:

```
import React, { useState, useEffect } from "react";

function Actors() {
  const url = "http://localhost:5297/Actor";
  const [actorData, setActorData] = useState([]);
  const [filteredActorData, setFilteredActorData] = useState([]);
  const [searchTerm, setSearchTerm] = useState(""); // Új állapot a kereséshez
  const [formData, setFormData] = useState({
    name: "",
    nationality: "",
    birthday: "",
    oscarAward: false,
    sex: "",
    kep: null, // Új mező a fájl kezelésére
  });
  const [editingActor, setEditingActor] = useState(null);
  const token = localStorage.getItem("authToken");
  const jog = Number(localStorage.getItem("authJog"));
```

A useEffect „hook” lekérése a szerver felé amivel lekérjük az adatbázisban lévő actor adatokat és ezt aszinkron módon tesszük. Továbbá elküldi az azonosításhoz szükséges tokent a szerver felé és kezeli a hibás lehívást:

```
useEffect(() => {
  fetchActors();
}, []);
```

```
const fetchActors = async () => {
  try {
    const response = await fetch(`${url}`, {
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`
      },
    });
    if (!response.ok) throw new Error("Hiba a színészek lekérésekor");
    const result = await response.json();
    setActorData(result.result ?? result);
    setFilteredActorData(result.result ?? result); // Beállítjuk a szűrt adatokat is
  } catch (error) {
    console.error(error.message);
  }
};
```

Hasonló useEffect alapon működő függvény ami a keresősávot támogatja és biztosítja hozzá az adatokat, megvizsgálja hogy van-e érték a kereső sávban majd leszűri az actor listát azzal az értékkel ami van benne, ha nincs akkor az eredeti listát adja vissza:

```
useEffect(() => {
  if (searchTerm === "") {
    setFilteredActorData(actorData);
  } else {
    setFilteredActorData(
      actorData.filter((actor) =>
        actor.name.toLowerCase().includes(searchTerm.toLowerCase())
      )
    );
  }
}, [searchTerm, actorData]);
```

Törlés függvény ami először megkérdezi hogy biztos vagy-e a dolgodban, majd elküldi a kijelölt elem ID-ját a törlés API végpontra. A kijelölt ID-t a megfelelő URL címmel tudja azonosítani a backend és a token segítségével. Küldi a DELETE metódust, törli a rögzítő listából a megszűntetett elemet és esetlegesen kezeli a hibákat a szerverlekéréskor.

```
const handleDelete = async (id) => {
  if (!window.confirm("Biztosan törlöd ezt a színészt?")) return;
  try {
    const response = await fetch(`${url}?id=${id}`, {
      method: "DELETE",
      headers: {
        Authorization: `Bearer ${token}`
      },
    });
    if (!response.ok) throw new Error("Hiba törlés közben");
    setActorData((prev) => prev.filter((actor) => actor.id !== id));
  } catch (error) {
    console.error(error.message);
  }
};
```

A szerkesztés gomb két részletben funkcionál, az első részben megnyitja a szerkesztő formot, megadja neki az actor létező adatait, kivéve a létező képet.

```
const handleEdit = (actor) => {
  setEditingActor(actor);
  setFormData({
    ...actor,
    kep: null, // Reseteljük a képfeltöltést szerkesztésnél
  });
};
```

A második része a „Mentés” funkciót biztosítja, ahol egy try, catch metóduson belül meghívja a form-nak az adatait, majd a PUT API végponton keresztül egy fetch lekéréssel elküldi a szervernek hogy változtassa meg. Ugyanez a függvény kezeli az editingactor értékétől

függően az új actorok felvitelét is. Ha igaz a szerkesztés(amit a handleEdit függvényben adsz meg) akkor PUT metódus lesz, ha hamis akkor pedig POST.

Végül meghívja a színészlekérő függvényt hogy frissüljön a renderelt lista vagy kezeli az esetleges hibákat.

```
const handleSave = async () => {
  try {
    const method = editingActor ? "PUT" : "POST";
    const endpoint = editingActor ? `${url}?id=${editingActor.id}` : url;

    const form = new FormData();
    form.append("Name", formData.name);
    form.append("Nationality", formData.nationality);
    form.append("Birthday", new Date(formData.birthday).toISOString().split("T")[0]);
    form.append("OscarAward", formData.oscarAward);
    form.append("Sex", formData.sex);

    if (formData.kep) {
      form.append("Kep", formData.kep); // Fájl hozzáadása
    }

    const response = await fetch(endpoint, {
      method,
      headers: {
        Authorization: `Bearer ${token}`
      },
      body: form,
    });
```

```
      if (!response.ok) throw new Error("Hiba mentés közben");

      setEditingActor(null);
      setFormData({
        name: "",
        nationality: "",
        birthday: "",
        oscarAward: false,
        sex: "",
        kep: null, // Reseteljük a képfeltöltést
      });
      fetchActors();
    } catch (error) {
      console.error(error.message);
    }
  };
```

Végül az Actor page(függvény) visszatérési értéke maga a kirajzolódó html-DOM kód, kezdve a form kinézettel. A form maga csak Admin kinézetben érhető el:

```
return (
  <div className="p-4">
    {jog === 1 && (
      <div className="mb-6 bg-gray-700 rounded shadow p-4">
        <h2 className="text-white font-bold text-lg font-bold mb-2">
          {editingActor ? "Színész szerkesztése" : "Új színész hozzáadása"}
        </h2>
        <input
          className="w-full p-2 border mb-2"
          type="text"
          name="name"
          placeholder="Név"
          value={formData.name}
          onChange={(e) => setFormData({ ...formData, name: e.target.value })}
        />
        <input
          className="w-full p-2 border mb-2"
          type="text"
          name="nationality"
          placeholder="Nemzetiség"
          value={formData.nationality}
          onChange={(e) =>
            setFormData({ ...formData, nationality: e.target.value })
          }
```

A kereső sávval:

```
{/* Keresősáv */}
<div className="mb-4">
  <input
    className="w-full p-2 border mb-4"
    type="text"
    placeholder="Keresés színész névre"
    value={searchTerm}
    onChange={(e) => setSearchTerm(e.target.value)}
  />
</div>

<div className="flex flex-wrap justify-center gap-3">
  {filteredActorData.map((actor) => (
    <div
      key={actor.id}
      className="max-w-sm rounded-lg overflow-hidden shadow-lg bg-gray-800 text-white"
```

A kártyákkal amin a színészek adatai megjelennek, a szerkesztés és törlés gombok csak admin joggal érhetőek el:

```
            {actor.profilKép && (
              <img
                src={actor.profilKép}
                alt={actor.name}
                className="w-full h-56 object-cover rounded-t-lg"
              />
            )}
          </div>
          <div className="p-4">
            <h2 className="text-xl font-bold">{actor.name}</h2>
            <p className="text-gray-400 text-sm">{actor.birthday.split("T")[0]}</p>
            <p className="text-gray-400 text-sm">{actor.nationality}</p>
            <p className="text-gray-400 text-sm">
              Oscar díjas: {actor.oscarAward ? "✅" : "❌"}
            </p>
            <p className="text-gray-400 text-sm">Nem: {actor.sex}</p>

            {jog === 1 && (
              <div className="flex gap-2 mt-3">
                <button
                  className="bg-blue-600 text-white px-3 py-1 rounded"
                  onClick={() => handleEdit(actor)}
                >
                  Szerkesztés
                </button>
```

```
                </button>
                <button
                  className="bg-red-600 text-white px-3 py-1 rounded"
                  onClick={() => handleDelete(actor.id)}
                >
                  Törlés
                </button>
              </div>
            )}
          </div>
        </div>
      ))}
    </div>
  </div>
  );
}

export default Actors;
```

Források

https://learn.microsoft.com/hu-hu/dotnet/csharp/fundamentals/tutorials/oop

https://learn.microsoft.com/hu-hu/shows/entity-framework-core-for-beginners/

https://www.apachefriends.org/faq_windows.html

https://www.geeksforgeeks.org/reactjs-router/

https://learn.microsoft.com/en-us/training/powerplatform/power-bi

https://tutorials.eu/data-transfer-objects-in-c-sharp/

https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm