Webes Alkalmazások Fejlesztése

1. beadandó/9.2. feladat
Gonda Dávid
BIXU0S
gonda.david18@gmail.com

Feladat

Mozi

Készítsünk egy mozi üzemeltető rendszert, amely alkalmas az előadások, illetve jegyvásárlások kezelésére.
1. részfeladat: a webes felületen keresztül a nézők tekinthetik meg a moziműsort, valamint rendelhetnek jegyeket.
• A főoldalon megjelenik a napi program, azaz mely filmeket mikor vetítik a moziban, valamint kiemelve az öt legfrissebb (legutoljára felvitt) film plakátja.
• A filmet kiválasztva megjelenik annak részletes leírása (rendező, főszereplők, hossz, szinopszis), plakátja, továbbá az összes előadás időpontja.
• Az időpontot kiválasztva lehetőség nyílik helyfoglalásra az adott előadásra. Ekkor a felhasználónak meg kell adnia a lefoglalandó ülések helyzetét (sor, illetve oszlop) egy, a mozitermet sematikusan ábrázoló grafikus felületen. Egyszerre legfeljebb 6 jegy foglalható, és természetesen csak a szabad helyek foglalhatóak (amelyek nem foglaltak, vagy eladottak). A felhasználónak ezen felül meg kell adnia teljes nevét, valamint telefonszámát, ezzel véglegesíti a foglalást.

2. részfeladat: az asztali grafikus felületet az alkalmazottak használják a mozipénztárakban az előadások meghirdetésére, illetve jegyek kiadására.
• Az alkalmazott bejelentkezhet (felhasználónév és jelszó megadásával) a programba, illetve kijelentkezhet.
• Új film felvitelekor ki kell tölteni a film adatait (cím, rendező, főszereplők, hossz, szinopszis), valamint feltölthetünk egy képet plakátként.
• Új előadás meghirdetéséhez a felhasználónak ki kell választania a termet, valamint a filmet, és az időpont megadásával hirdetheti meg az előadást. A meghirdetéskor ügyelni kell arra, hogy az előadás ne ütközzön más előadásokkal az adott teremben (figyelembe véve a kezdés időpontját, illetve a film hosszát), illetve két előadás között legalább 15 percnek kell eltelnie a takarítás végett.
• A jegyvásárláshoz ki kell választani a filmet és az előadást. Ezt követően listázódnak a helyek (sor, oszlop, státusz). A szabad, illetve foglalt helyek eladhatóak, illetve a foglalt helyeket kiválasztva meg lehet tekinteni a foglaló adatait (név, telefonszám).

Az adatbázis az alábbi adatokat tárolja:
• filmek (cím, rendező, szinopszis, hossz, plakát, bevitel dátuma);
• termek (név, sorok száma, oszlopok száma);
• előadások (film, kezdő időpont, terem);

• helyek (előadás, terem, sor, oszlop, státusz , foglaló neve, foglaló telefonszáma);
• alkalmazottak (teljes név, felhasználónév, jelszó).

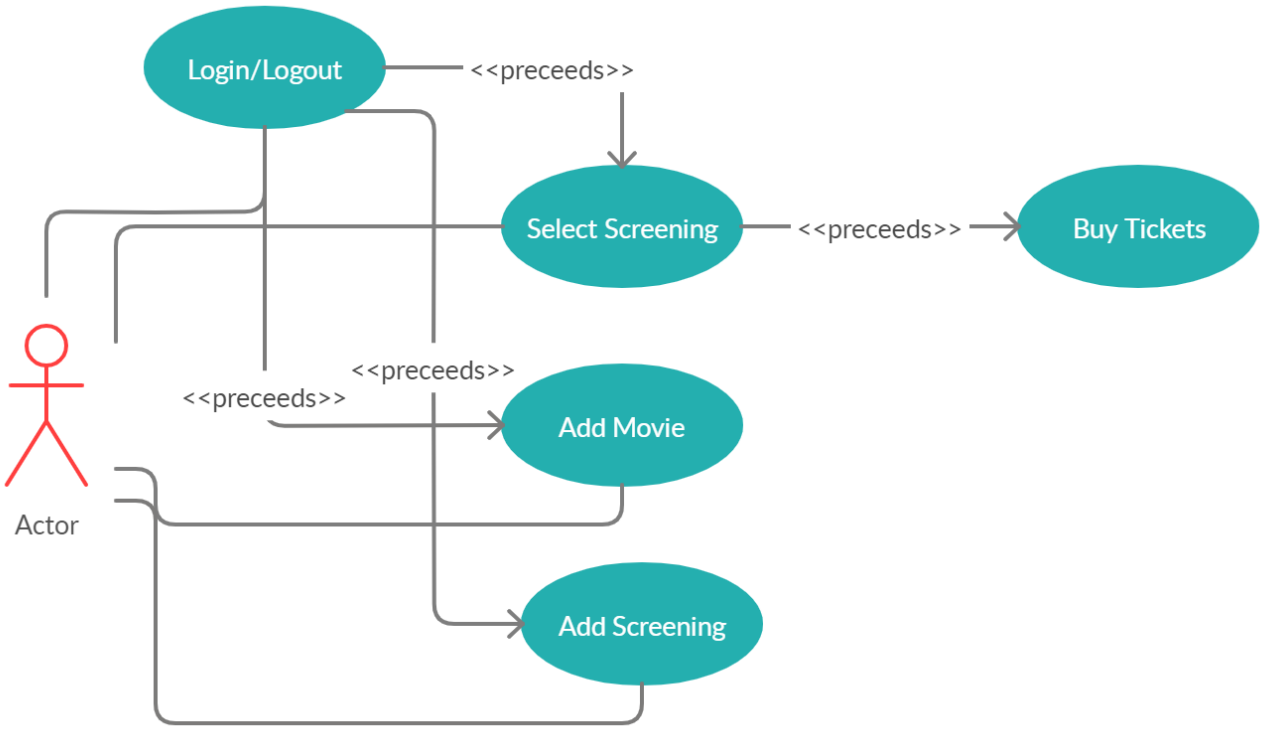
Elemzés

- Az alkalmazást *WPF*-ben, MVVM architektúrában valósítjuk meg. 5 ablakból fog állni: egy **Bejelentkező** ablakból, itt lehet bejelentkezni az alkalmazásba; egy **Központi** ablakból, itt soroljuk fel a vetítéseket, itt lehet kezdeményezni rajtuk a jegyvásárlást, valamint itt lehet kezdeményezni az adatbázishoz való film, illetve vetítés hozzáadását; egy **Jegyvásárló** ablakból, nevéből adódóan itt lehet vásárolni jegyeket; egy **Film hozzáadó** ablakból, itt lehet további filmeket hozzáadni az adatbázishoz; végül pedig egy **Előadás hozzáadó** ablakból, nevéből sejthető, hogy itt mit csinálhatunk.
- A vetítéseket, termeket, üléseket, és filmeket egy adatbázisban fogjuk tárolni.
- Az adatbázis 5 táblából fog állni: **Seats**, itt tároljuk vetítésekre bontva a székeket, amiknek megadjuk a vetítés azonosítóját, a terem azonosítóját, a teremben való elhelyezkedésüket, az aktuális állapotukat (*Szabad, Foglalt, Eladva*), a foglaló nevét és számát, a két utóbbi kezdedben *NULL,* ezek cask foglalás esetén adhatóak meg; **Screenings**, itt tároljuk a film azonosítóját, a kezdőidőpontot, valamint a terem azonosítóját; **Movies,** itt tároljuk a film címét, a rendező nevét, egy rövid szinopszist, a film hosszát, a plakátot, valamint a film bevitelének dátumát; **Rooms,** itt tároljuk a termek nevét, a sorok/oszlopok számát; **Employee**, itt tároljuk a felhasználókat, *IdentityUser*-ből származtatott.
- Az adatbáziselérést egy **API** segítségével valósítjuk meg, itt az összes tipikus lekérdezést megvalósítjuk, illetve ahol szükséges, a *Create*-et és az *Update*-et is.

-

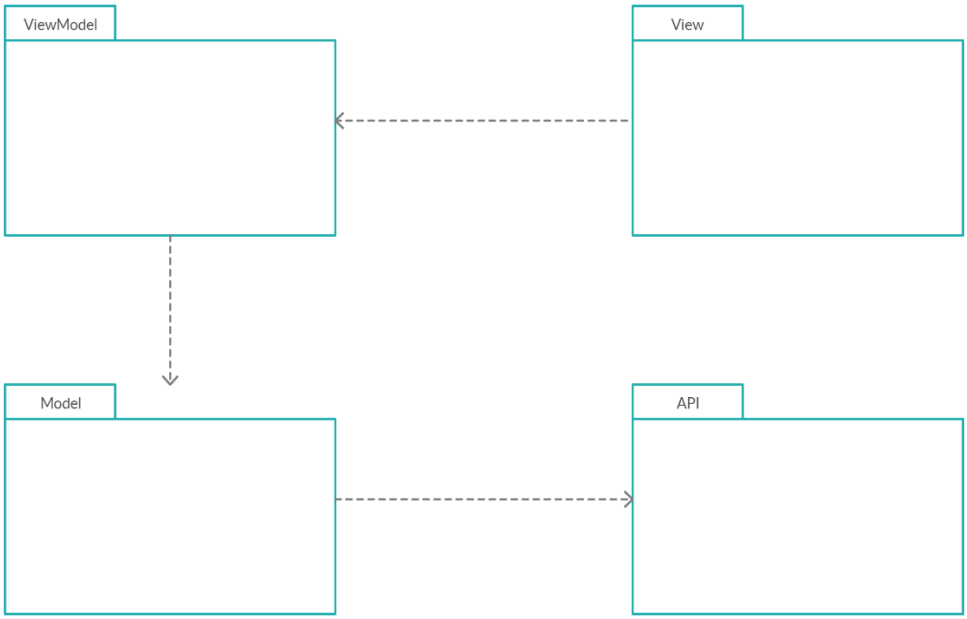
Felhasználói esetek

- A felhasználónak először be kell jelentkeznie, ameddig ezt nem teszi meg nem csinálhat semmit.
- Bejelentkezés után kiválaszthat egy vetítést, hozzáadthat egy filmet az adatbázishoz, vagy hozzáadhat egy vetítést az adatbázishoz.
- Vetítés kiválasztása esetén kijelölheti a nem eladott helyeket, amiket ezután eladhat.



1. Use Case diagram

Osztálydiagram

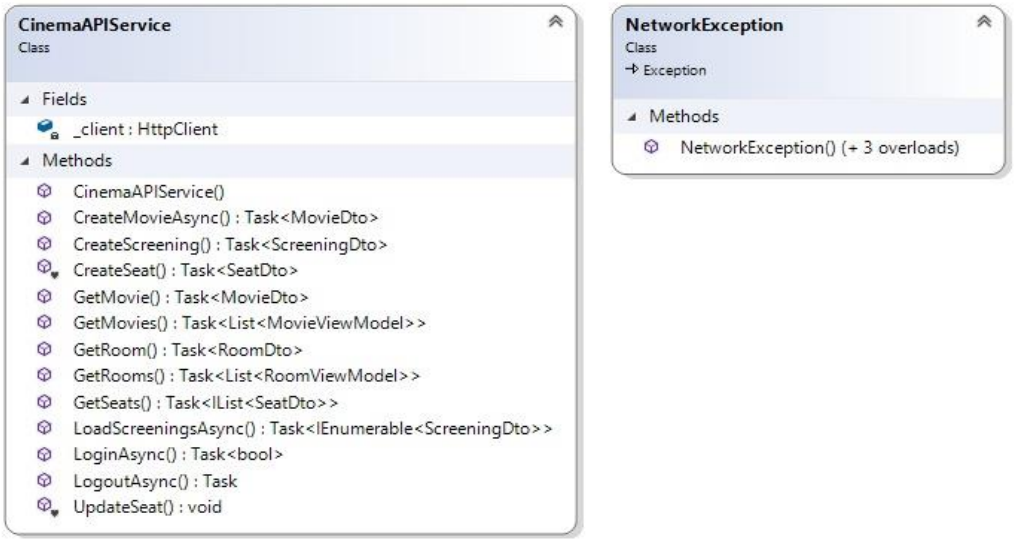


2. Komponens diagram

A program 3 nagyobb egységből áll, egy *View*ból, egy *ViewModel*ből, és egy *Model*ból.

Ezeken kívül még hozzávesszük az **API** projektünket is, mivel a *Model*-ünk ez alapján éri el az adatbázist.

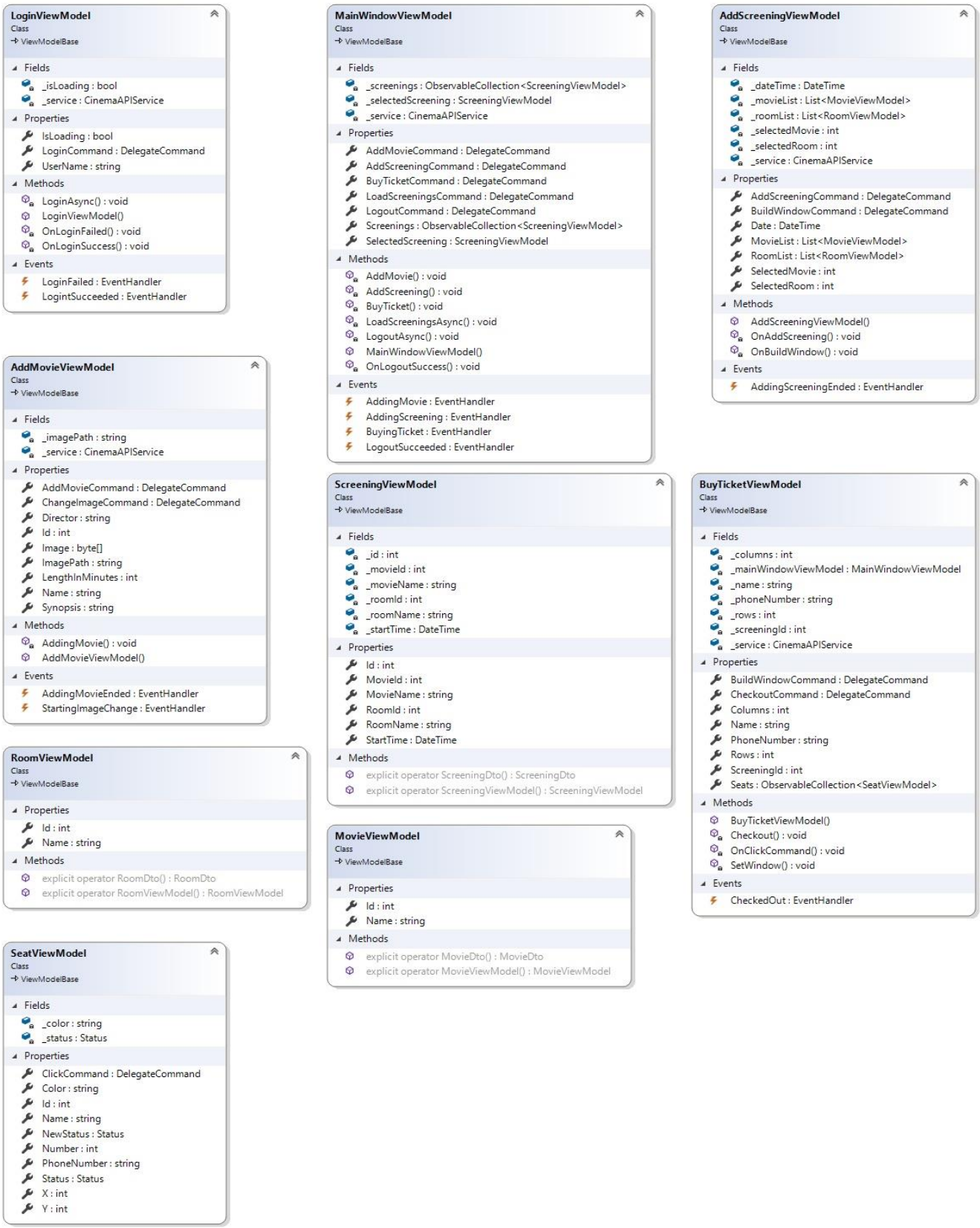
- **Model:**



3. Model class diagram

A *Model*-ünk valósítja meg az **API-**hoz való elérésünket.
A szükséges lekérdezéseket, frissítéseket, létrehozásokat tartalmazza.

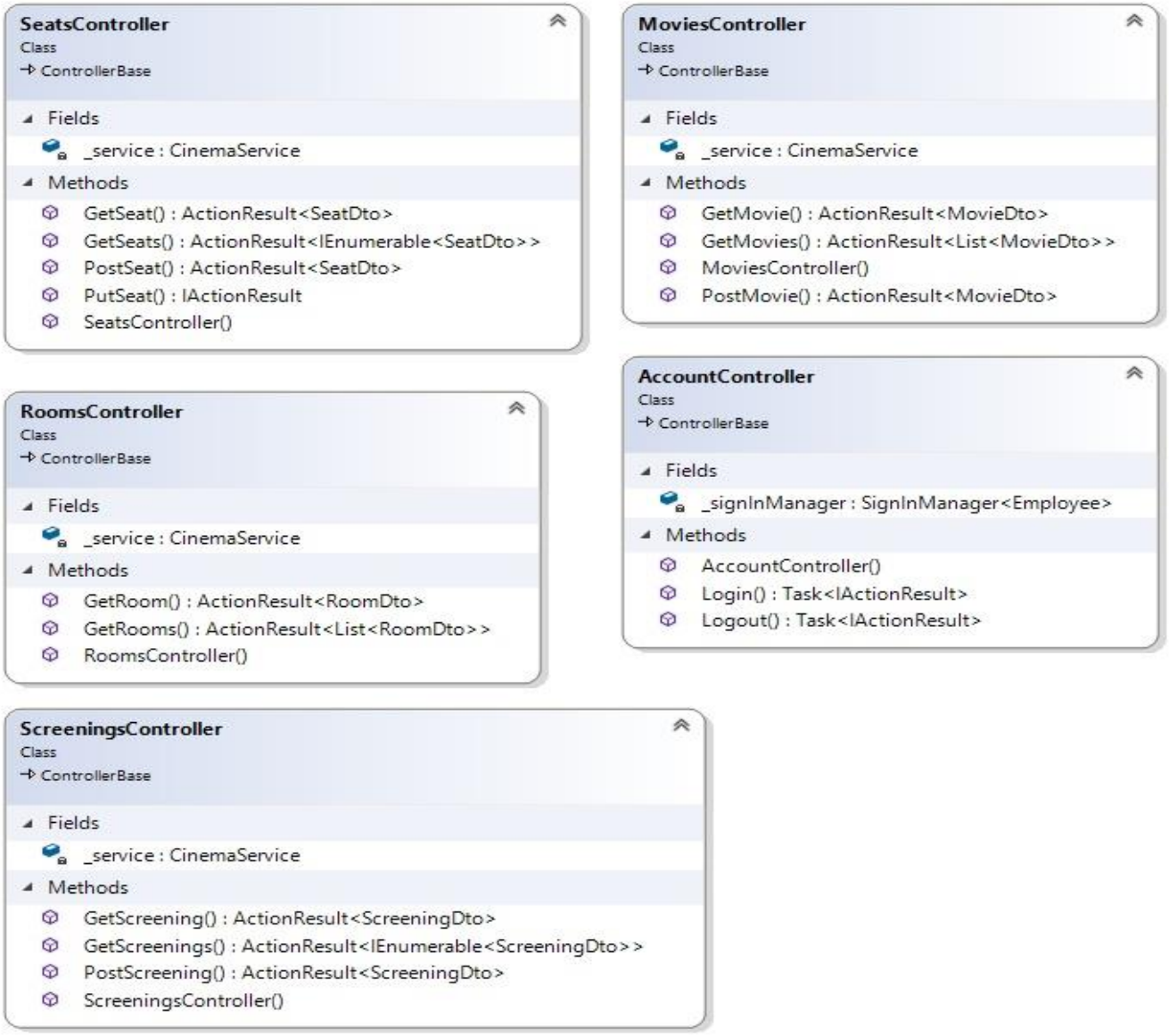
- **ViewModel:**



4. ViewModel class diagram

A nézetmodellek, amelyek a megjelenítést segítik a *View*-nak.
A fontosabb osztályok innen az *AddMovieViewModel, AddScreenViewModel a MainWindowViewModel,* és a *BuyTicketViewModel.*
Az *AddMovieViewModel* valósítja meg a film adatbázishoz való adását; az *AddScreenViewModel* valósítja meg a vetítés adatbázishoz való adását; a MainWindowViewModel megjeleníti az összes vetítést, továbbá itt lehet kiválasztani, hogy melyik vetítésre szeretnénk jegyet vásárolni, és hogy akarunk e filmet/vetítést adni az adatbázishoz; végül pedig a *BuyTicketViewModel* valósítja meg a jegyvásárlást, továbbá ha az adatbázisban nem szerepelnének még a vetítéshez tartozó székek, ezeket is hozzáadja az adatbázishoz.

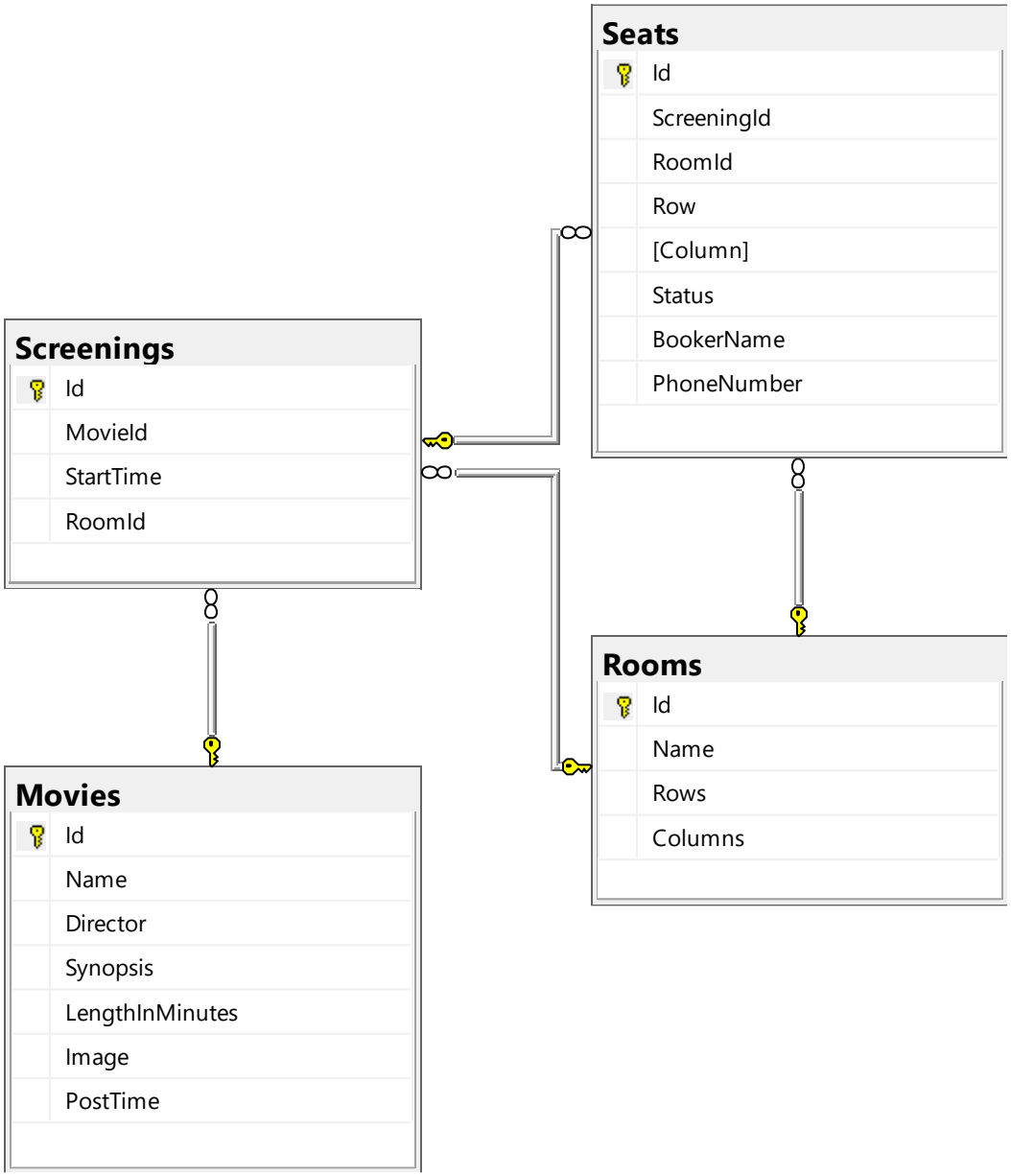
- **API:**



***5**. API class diagram*

Az adatbáziselérést valósítja meg.

Adatbázis



6. Adatbázis entity relationship diagram

Az adatbázisunk négy táblából áll, továbbá az *IdentityUser* által generált táblákból, ezeket a nagy számuk miatt inkább nem jelenítjük meg a dokumentációban.