Webes Alkalmazások Fejlesztése

1. beadandó/9.1. feladat  
Gonda Dávid  
BIXU0S  
[gonda.david18@gmail.com](mailto: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.

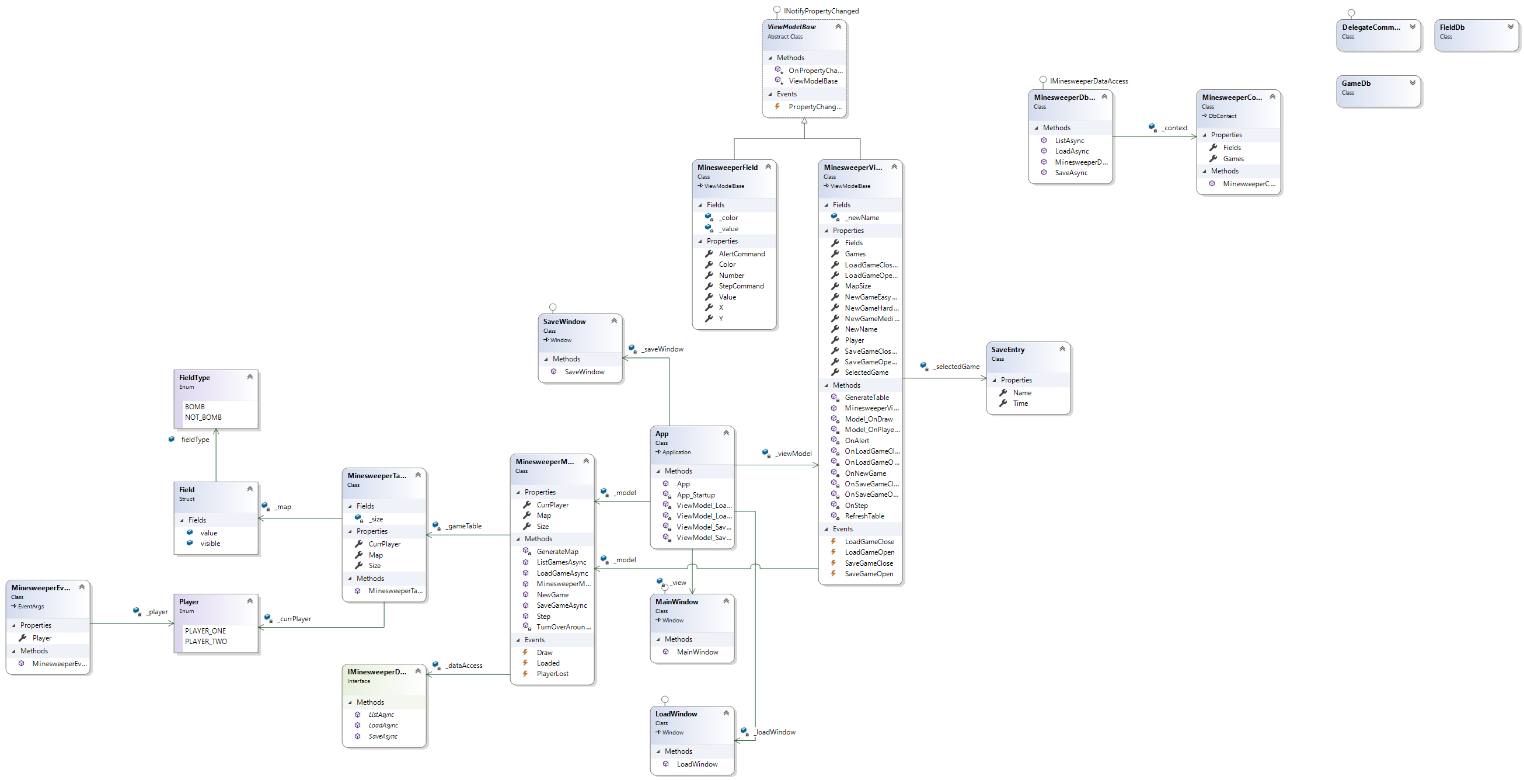
# *2. részfeladat*: az asztali grafikus felületet az alkalmazottak használják a mozipénztárakban az előadások meghirdetésére, illetve jegyek kiadására. • Az alkalmazott bejelentkezhet (felhasználónév és jelszó megadásával) a programba, illetve kijelentkezhet. • Új film felvitelekor ki kell tölteni a 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 oldal 3 fő oldalból fog állni: egy **Főoldalből**, ahol kilistázzuk a napi programokat; egy **Film** oldalból, ahol a kiválasztott film adatait, illetve az előadások időpontját/termét lehet látni/kiválasztani; és egy **Foglalási** odlalból, itt megjelenítjük a termet, amin kattintással lehet kiválasztani a foglalt helyeket, majd pedig név és telefonszám megadásával után lehet megerősíteni a foglalást.
* A feladatot *ASP.NET Core*-ban valósítjuk meg, *MVC*-ben.
* A foglalás után átirányítjuk a felhasználót egy **Siker** oldalra, ahonnan 4 másodperc után visszairányítjuk a főoldalra.
* A vetítéseket, termeket, üléseket, és filmeket egy adatbázisban fogjuk tárolni.
* Az adatbázis számunkra 4 fontosabb 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.
* Még egy táblát kell megemlítenünk, mégpedig az **Employees** táblát, de erről nem ejtenék most szót, mivel ez csak a következő részfeladatban less aktuális.

# Osztálydiagram



A program 4 nagyobb egységből áll, egy *model*ből, egy *viewmodel*ből, egy *view*ból, és egy *persistence* rétegből.

* **Model:**Ez a réteg felel a játék üzleti logikájáért. Főbb metódusai a *Step* (egy játéklépést hajt végre)*,* a *NewGame* (új játékot generál térképnagyság szerint), és a *GenerateMap* (legenerálja a .pályát).
* **ViewModel:**A *view*ot a *model*el összekötő egység, leginkább paramétereket tartalmaz a *view* számára a *model*ből, de tartalmazza a pálya legenerálását is vizuálisan, az inputok hatását is kezeli, összeköti a *model* metódusaival, illetve a job egérgomb hatására jelzést rak le a *model*től függetlenül.
* **View:**A grafikust megjelenítést leíró egység. Áll egy mentő-/betöltő-/főablakból. Dolga a grafikus megjelenítésen túl a különböző **commandok, változók** nevének megadása.
* **Persistence:**A mentés/betöltést végző egység. Adatbázissal (Entity Framework) valósítjuk meg, aszinkron módon. Több mentett állást is lehet tárolni, amik közül aztán lehet választani betöltéskor.

# Tesztelés

* A modell funkcionalitását egységtesztek segítségével ellenőriztük a **MinesweeperTest** osztályban. Ehhez létrehoztunk egy mock perzisztenciát (egy mock táblával együtt).
* Az alábbi tesztesetek kerültek megvalósításra:
  + **MinesweeperModelNewGameTest:**Új játék indításának az ellenőrzése.
  + **MinesweeperModelStepTest:**A lépésnek, annak hatásainak ellenőrzése.
  + **MinesweeperModelTurnOverAroundZeroTest:**Nullára való lépés után a mező körül lévő mezők felfedésének ellenőrzése, több egymás melletti nullával.
  + **MinesweeperModelGenerateMapTest:**Tábla generálásának az ellenőrzése.