# BookEr Asztali alkalmazás és WebApi

**Készítette:**

Nagy Dániel

**Feladat:**

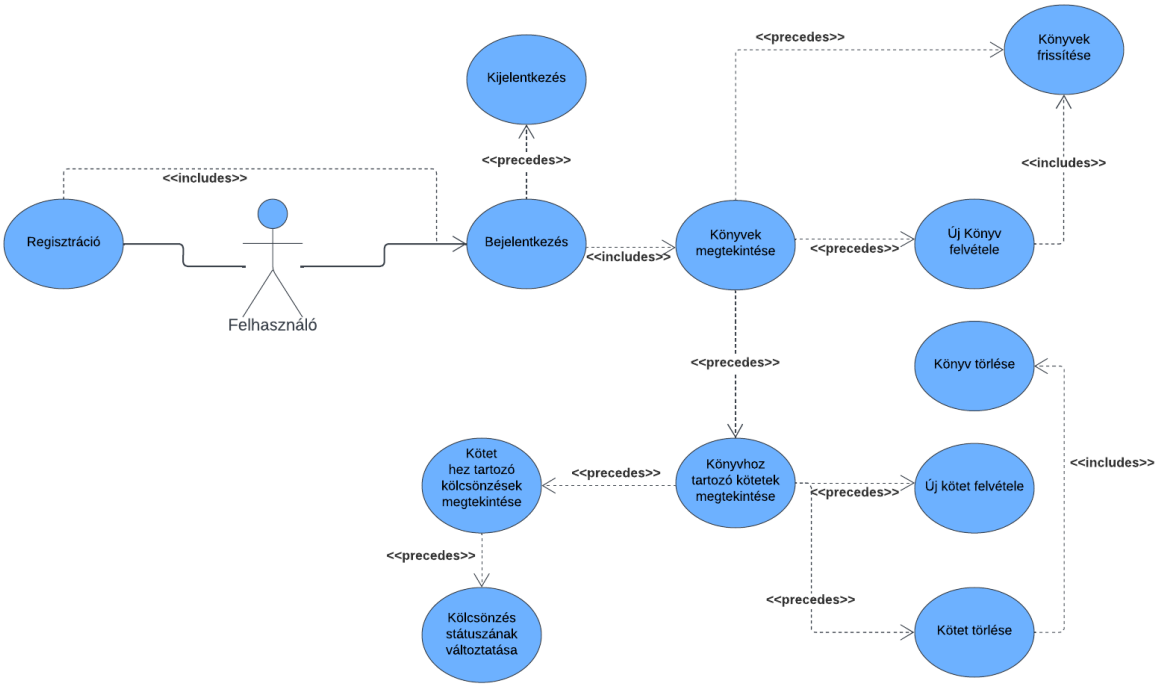
A könyvtárosok az asztali grafikus felületen keresztül adminisztrálhatják a könyveket és a kölcsönzéseket.

- A könyvtáros bejelentkezhet (felhasználónév és jelszó megadásával) a programba, illetve kijelentkezhet; a további funkcionalitások csak bejelentkezett állapotban elérhetőek.
- Az alkalmazás listázza a könyveket, valamint a hozzájuk tartozó köteteket. Lehetőség van új könyv, illetve kötet rögzítésére.
- A könyvtáros selejtezhet egy kötetet, de csak akkor, ha nincsen aktuálisan kikölcsönözve. Az esedékes jövőbeni előjegyzések törlésre kerülnek és az adott kötet továbbá nem lesz kölcsönözhető.
- Az alkalmazás listázza az aktív kölcsönzéseket és a jövőben esedékes előjegyzéseket. A könyvtáros egy aktív kölcsönzést inaktívvá tehet (visszavitték a könyvet), valamint egy inaktív előjegyzést aktív kölcsönzésnek jelölhet (elvitték a könyvet). Egy kölcsönzés státuszának változtatása nincs a tervezett kezdő és befejező naphoz kötve (gondolva pl. a késedelmes visszavitelre), azonban egy kötetnek egyszerre legfeljebb egy aktív kölcsönzése lehet.

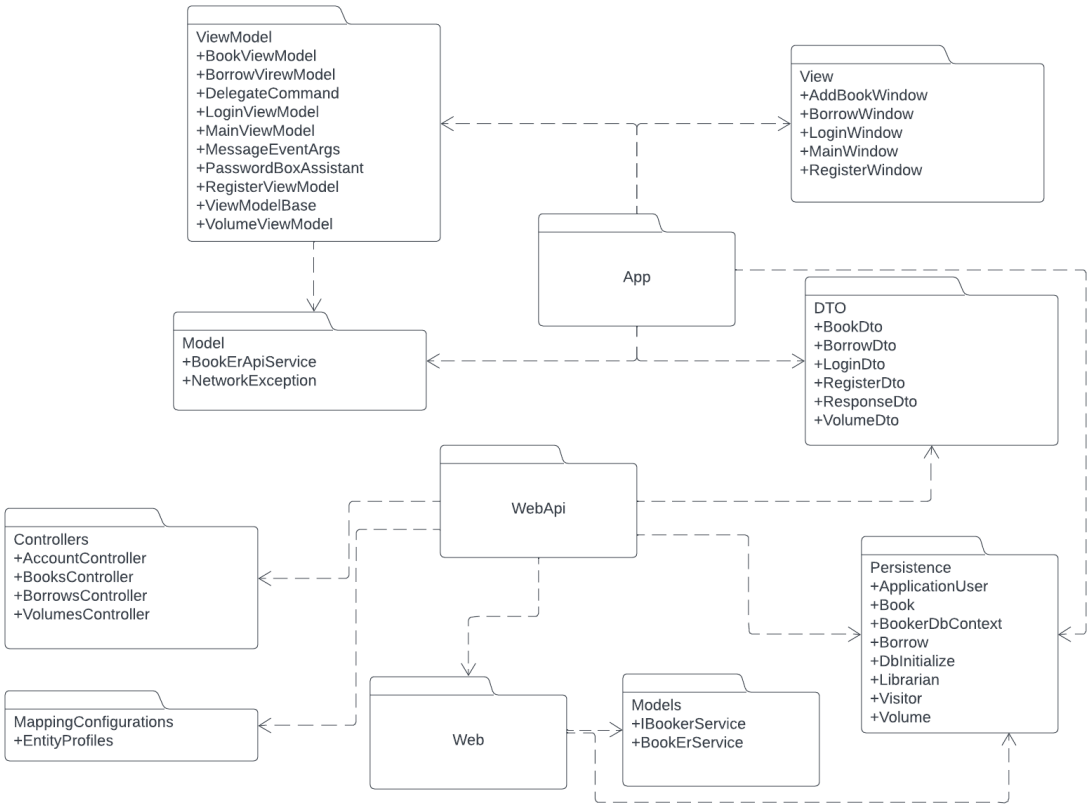
**Feladat Elemzése:**

Az első részfaladat megoldását refaktorálva külön projektekbe szervezzük a különböző komponenseket. Az adatbázis code-first definiálásáért a persistence projekt lesz a felelős. Az asztali kliens alkalmazást WPF keretrendszerben írjuk, ami egy, az előző részfeladatban látott megoldáson alapuló WebApi-hoz tud kéréseket intézni. A WebApi teremti meg a kapcsolatot az adatbázis, és a kliens között. A WebApi és az asztali alkalmazás közötti adatátvitelre DTO (Data Transfer Object) osztályokat használunk.

## UseCase Diagram

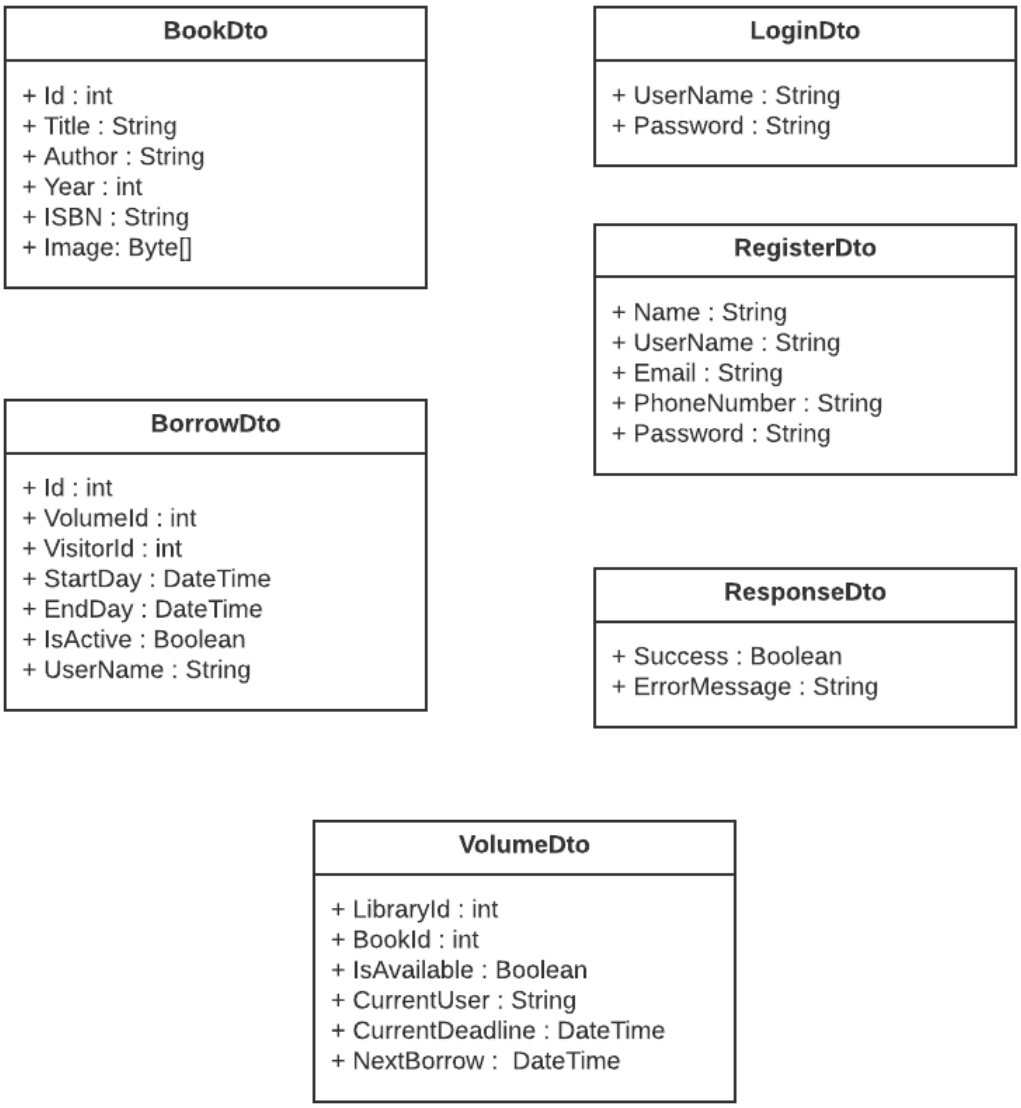



## Csomag Diagram:



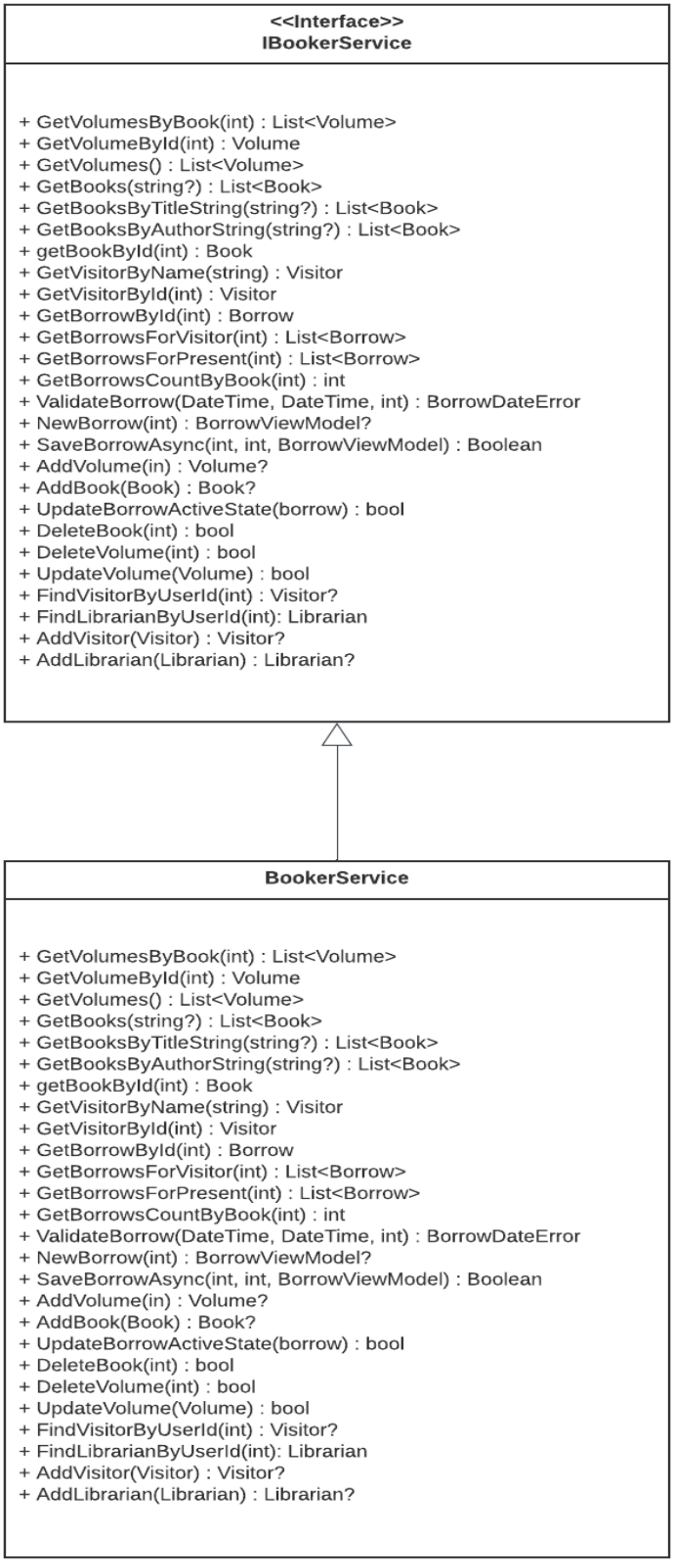
## DTO Projekt (BookEr.Dto)

A data transfer object-ek segítségével az asztali kliens és a WebApi közötti információközlés megvalósítható. A DTO-k többnyire az adatbázis tábláit reprezentálják a navigációs propertyk nélkül, így elkerülve a körkörös hivatkozásokat. A DTO-k igény szerint tartalmazhatnak plusz adattagokat is amennyiben ezek a megjelenítéshez kellenek.

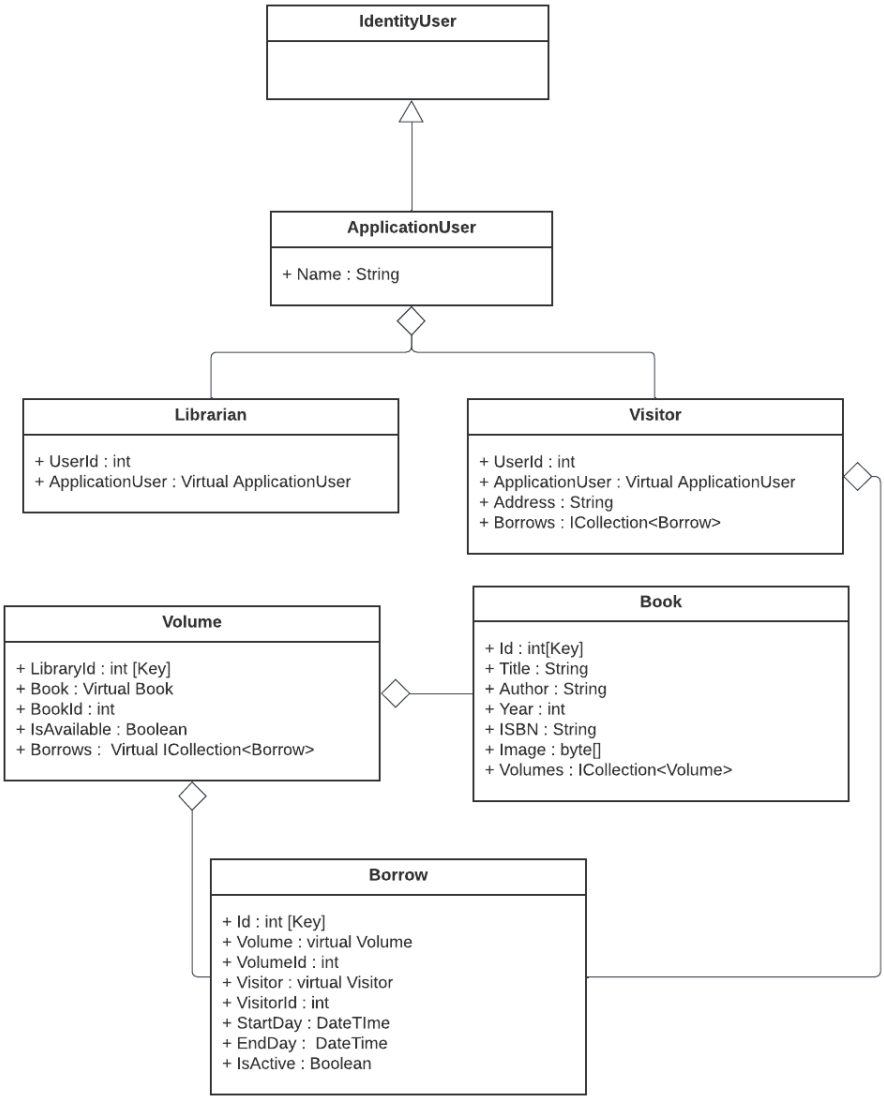


## Services

A Service osztály az adatbázis kezelésért felel.  A service által definiált metódusok segítségével tudja a többi komponens manipulálni az adatbázist.



## Adatbázis



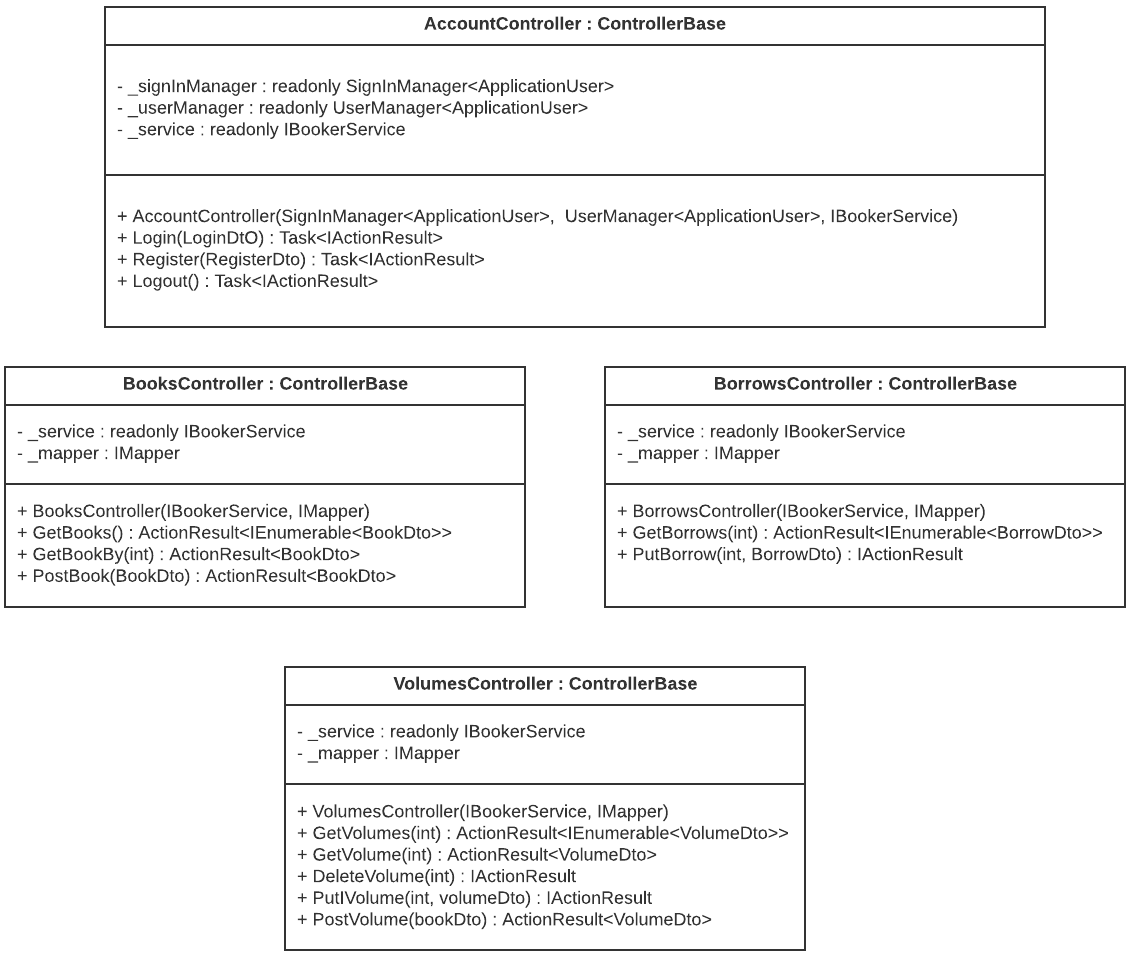

## WebApi projekt (BookEr.WebApi)

A szerver és kliens közötti kommunikációért egy HTTP kéréseket teljesítő WebApi projekt felel. Az 5. ábrán láthatóak a WebApi projekt kontrollerei.

Az alkalmazás WebAPI kontrollerei implementálják a ControllerBase nevű osztályt. Ezek kezelik a HTTP kéréseket és generálják rá a válaszokat. Az osztályokban definiált metódusok (végpontok) fogadják a különböző HTTP kéréseket (GET, POST, PUT, DELTE). A metódusokat különböző attribútumokkal látjuk el, amelyek meghatározzák a végpont útvonalát, a paramétereket, és az autorizációt. HTTP kérés érkezésekor az ASP.NET keretrendszer meghívja a megfelelő metódust a kontrollerben. Bizonyos esetben paraméterek is érkeznek. Válaszként mindig egy HTTP státusz kódot

küldenek a program kontrollereinek metódusai, amelyekhez bizonyos esetekben tartalmat is csatolnak (pl. hibaüzenet).

### WebApi Kontroller osztálydiagram:




### MappingConfigurations:

A MappingConfigurations könyvtárban található az EntityProfiles osztály, ami az adatbázis objektumok és a hozzájuk tartozó data transfer objectek között biztosítja a konverziókat. Ilyen konverziós osztályokat minden DTO típushoz implementálunk (Borrow, Book, Volume) az oda-vissza konverziókhoz.

## Asztali alkalmazás (BookEr.Desktop)

Az App.config fájlban beállítjuk a baseAddress értékét a használni kívánt elérési útvonalra, hogy ha megváltozik a cím vagy a port, akkor könnyen módosítható legyen.

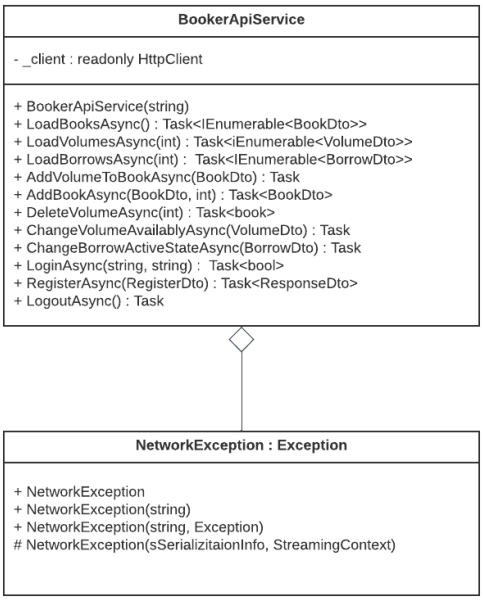
Az asztali alkalmazás nézet-modell-nézetmodell (model-view-viewmodel MVVM) architektúrában van megvalósítva. A modell az üzleti logikát (ezesetben a webapival való kommunikációt) valósítja meg, míg a nézet az ablakok és az azokban található elemek megjelenítéséért felel. A nézetmodell a megjelenítésért felelős logikát valósítja meg.

Az MVVM architektúra segít elkülöníteni az alkalmazás grafikus felületét (nézet) és a hátterében futó üzleti logikát (modell). Ez javítja a kód olvashatóságát és karbantarthatóságát, valamint könnyen újra fel lehet azt használni (ugyanahhoz a modell és nézetmodell rétegekhez több grafikus felület is megvalósítható). Az MVVM lehetővé teszi az adatkötés használatát a nézet és a modell között a közvetett adatkötés (data binding) révén. Ez azt jelenti, hogy a nézet automatikusan frissül, amikor a modell adatai változnak, és fordítva.

## Modell réteg

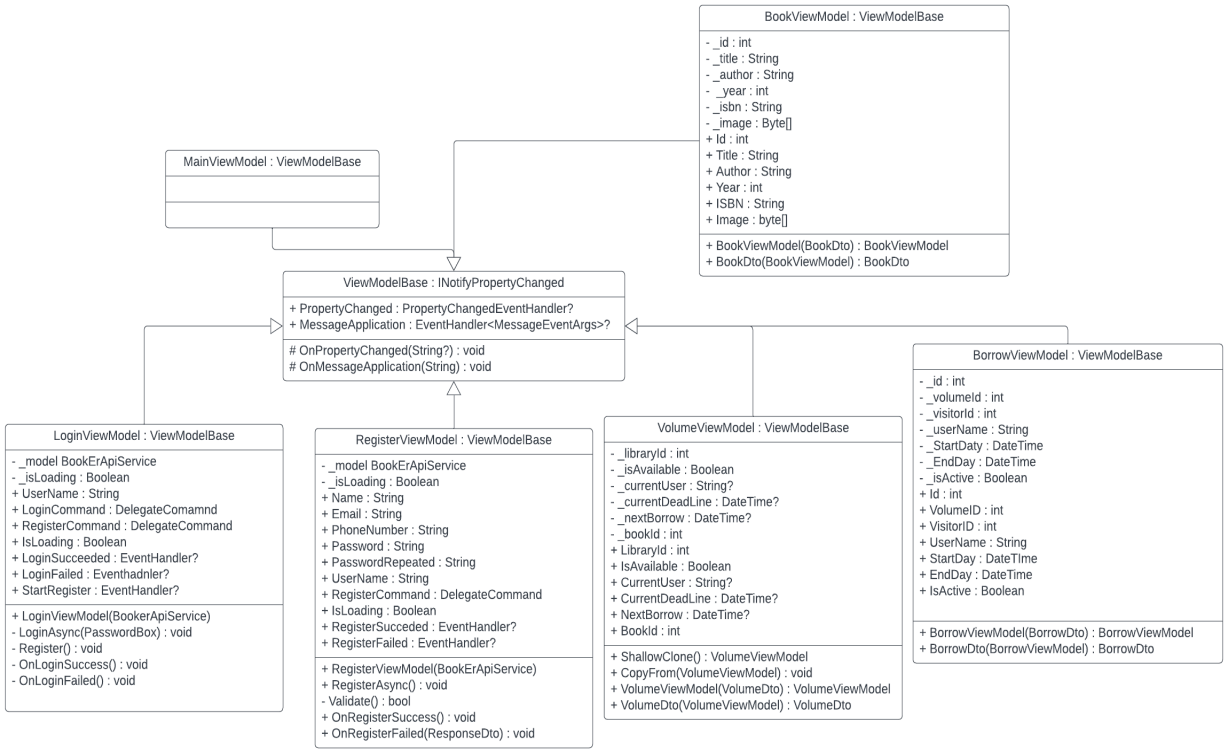
A BookerApiService osztály példányosít egy HttpClient-et és létrehozza a kapcsolatot a szerverrel a baseAddressen keresztül. Feladata, hogy továbbítsa a kliens kéréseit a szerver felé. Ezt aszinkron módon valósítja meg.

Ha a kommunikáció során kivétel keletkezik, akkor egy általunk definiált kivételt dobunk (Network Exception), amelynek az üzenetében továbbítjuk a hibakódot.

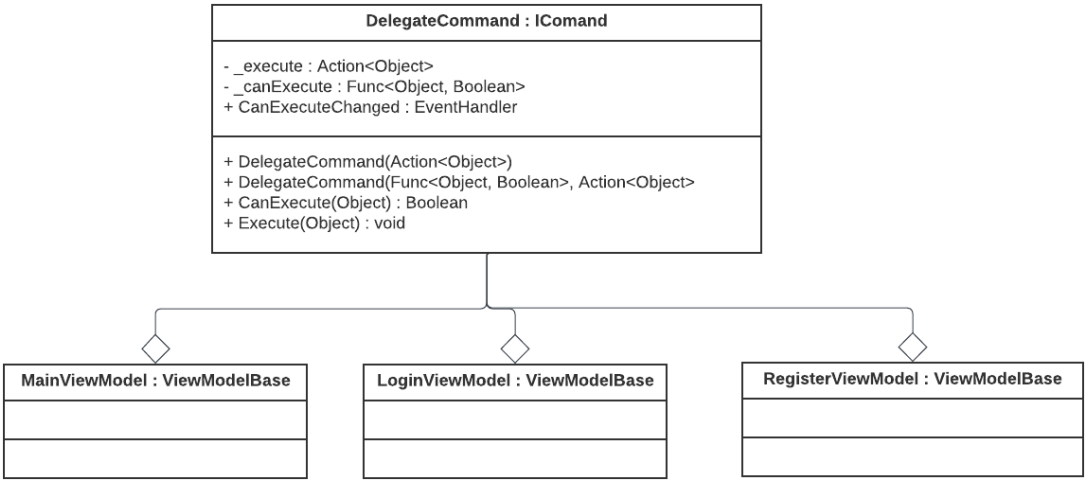


## Nézetmodell réteg

A nézetmodell megvalósításához felhasználunk egy általános utasítás (**DelegateCommand**), valamint egy ős változásjelző (**ViewModelBase**) osztályt. Ezeknek a kötései, illetve kiemelve a MainViewModel a következő 3 ábrán láthatóak. A MainViewModel osztály felel a fő ablak működéséért és összeköti az összes többi nézettel.




## DelegateCommand kötései




## Nézet réteg

A nézet 5 ablakot tartalmaz.

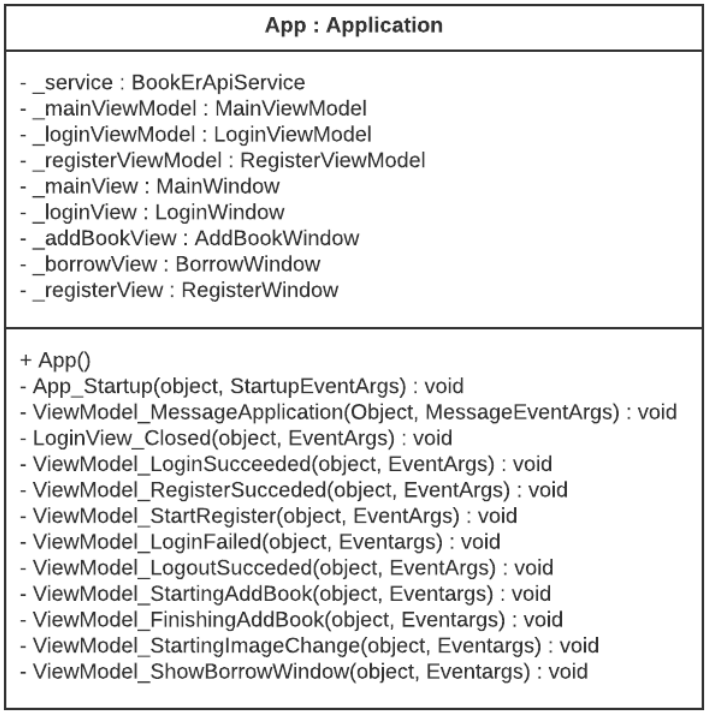
Az applikáció indítása után a LoginWindow fogadja a felhasználót. Itt lehetősége van bejelentkezni a többi ablak eléréséhez, vagy regisztrálni. Utóbbi Esetben át lesz irányítva a RegisterWindow nézetre.

Sikeres bejelentkezés, regisztráció után megjelenik a MainWindow, ahol a program legtöbb funkciója elérhető. A Felhasználó itt látja az aktuálisan adatbázisban lévő könyveket. Tud új könyvet/kötetet felvenni és törölni. Új könyv hozzáadásakor az AddBookWindow fog megjelenni, itt tudjuk megadni a szükséges adatokat.

Ha egy kötetnek vannak bejegyzett kölcsönzései, akkor ezeket a BorrowWindow segítségével tudjuk megnézni, illetve módosítani egyes foglalások aktív státuszát.

## Applikáció osztály

Az App osztály feladata az egyes rétegek példányosítása (App_Startup), összekötése, a nézetmodell, valamint a modell eseményeinek lekezelése, és ezáltal az üzleti logika, az adatkezelés, valamint a nézetek szabályozása. Az App osztály osztálydiagramja a következő ábrán látható.



## Tesztelés

A WebApi réteg kontroller osztályainak metódusai **xUnit** típusú projektben lettek tesztelve. A teszteket a ControllersTest osztály valósítja meg. A teszteléshez „in-memory” adatbázist használunk, amit minden teszteset elején feltöltünk, a végén pedig törlünk.

**BooksController:**

**GetBooksTest**: Az összes könyv lekérdezése. Ellenőrizzük, hogy a kapott lista számossága egyezik-e az álltalunk felvett könyvek számával

**GetBookByIDTest:** Három különböző id-val lekérünk egy-egy könyvet, majd ellenőrizzük, hogy az eredmény id-ja megegyezik-e az álltalunk adottal.

**GetInvalidBookTest:** Érvénytelen id-val próbálunk lekérni egy nemlétező könyvet, majd ellenőrizzük, hogy a szerver „Not Found” al válaszolt-e.

**PostBookTest**: Létrehozunk egy új könyvet, hozzáadjuk az adatbázishoz, majd lekérdezzük id alapján és ellenőrizzük az adatokat.

**BorrowsController:**

**GetBorrowsTest:** Lekérdezzük több különböző kötethez tartozó foglalásokat és ellenőrizzük ezek számoságát.

**GetBorrowsInvalidTest:** Nemlétező kötethez kérünk le kölcsönzéseket.

**ChangeBorrowActiveStateTest:** Lekérünk egy kölcsönzést és a hozzá tartozó kötetet. megváltoztatjuk a kölcsönzés státuszát majd újra lekérve az adatokat ellenőrizzük a kölcsönzés, és a kötet aktuális státuszát.

**GetBorrowsInvalidTest:** Nemlétező kötethez kérünk le kölcsönzéseket.

**ChangeBorrowActiveStateInvalidTest:** Megpróbálunk egy olyan kötethez tartozó kölcsönzést aktívvá tenni, aminek már van aktív kölcsönzése. Ellenőrizzük, hogy a jó státuszkódot kapjuk-e vissza.

**VolumesController:**

**GetVolumesTest:** Lekérdezzük több különböző könyvhöz tartozó köteteket és ellenőrizzük ezek számoságát.

**GetVolumesInvalidTest:** Nemlétező könyvhöz kérünk le kölcsönzéseket.

**GetVolumesByIdTest:** 10 különböző id-val kérünk le kötetet az adatbázisból, majd összehasonlítjuk az id-kat.

**DeleteVolumeTest:** Kitörlünk egy kötetet, majd megpróbáljuk lekérni újra az adatbázisból. Ellenőtizzük, hogy „Not Found”-ot kapunk-e.

**DeleteBookTest:** Lekérünk egy könyvet, aminek egyetlen kötete van, kitöröljük azt, majd leellenőrizzük, hogy törlődött-e a könyv.

**DeleteVolumeInvalidTest:** Megpróbálunk aktív kölcsönzéssel rendelkező kötetet törölni, majd ellenőrizzük, hogy helyes státuszkódot kapunk-e vissza.

**PutVolumeTest:** Megváltoztatjuk az egyik kötet elérhető státuszát, majd újra lekérdezve azt, ellenőrizzük, hogy valóban módosult-e.

**PostVolumeTest:** Lekérünk egy könyvhöz tartozó köteteket. Majd hozzáadunk egy új kötetet a könyvhöz. Végül ellenőrizzük hogy újból lekérve a kötetek száma növekedett-e.