

BookEr Asztali alkalmazás és WebApi

Készítette:

Nagy Dániel

Feladat:

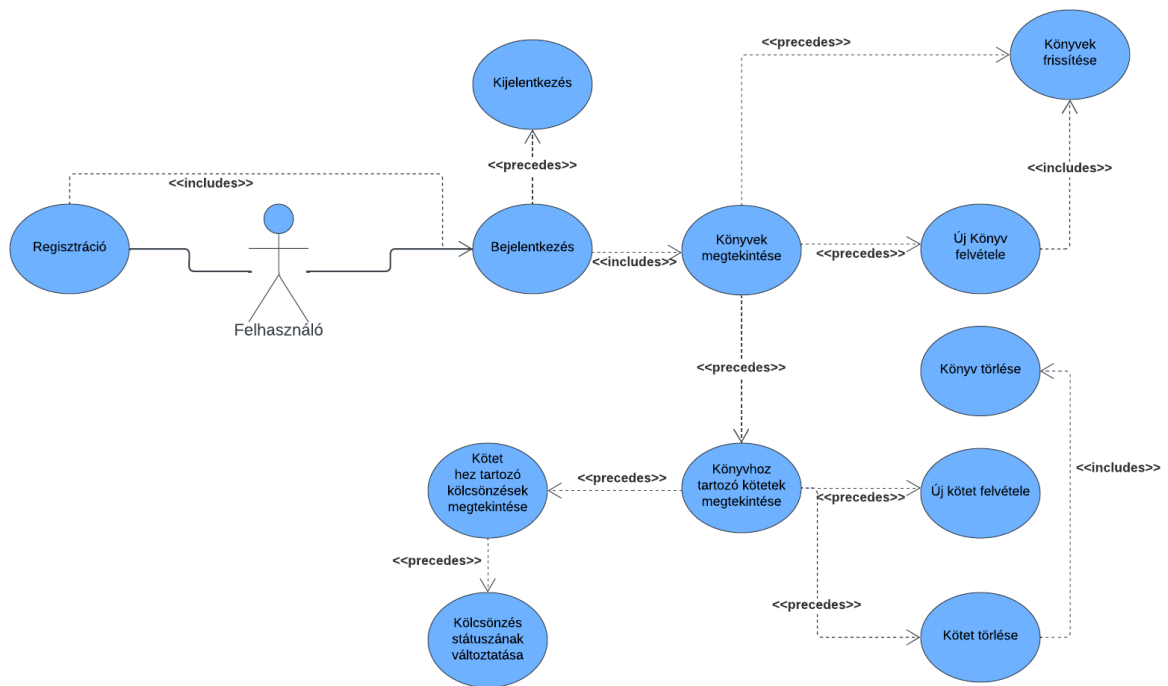
A könyvtárosok az asztali grafikus felületen keresztül adminisztrálhatják a könyveket és a kölcsönzéseket.

- A könyvtáros bejelentkezhet (felhasználónév és jelszó megadásával) a programba, illetve kijelentkezhet; a további funkciók csak bejelentkezett állapotban elérhetők.
- 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.

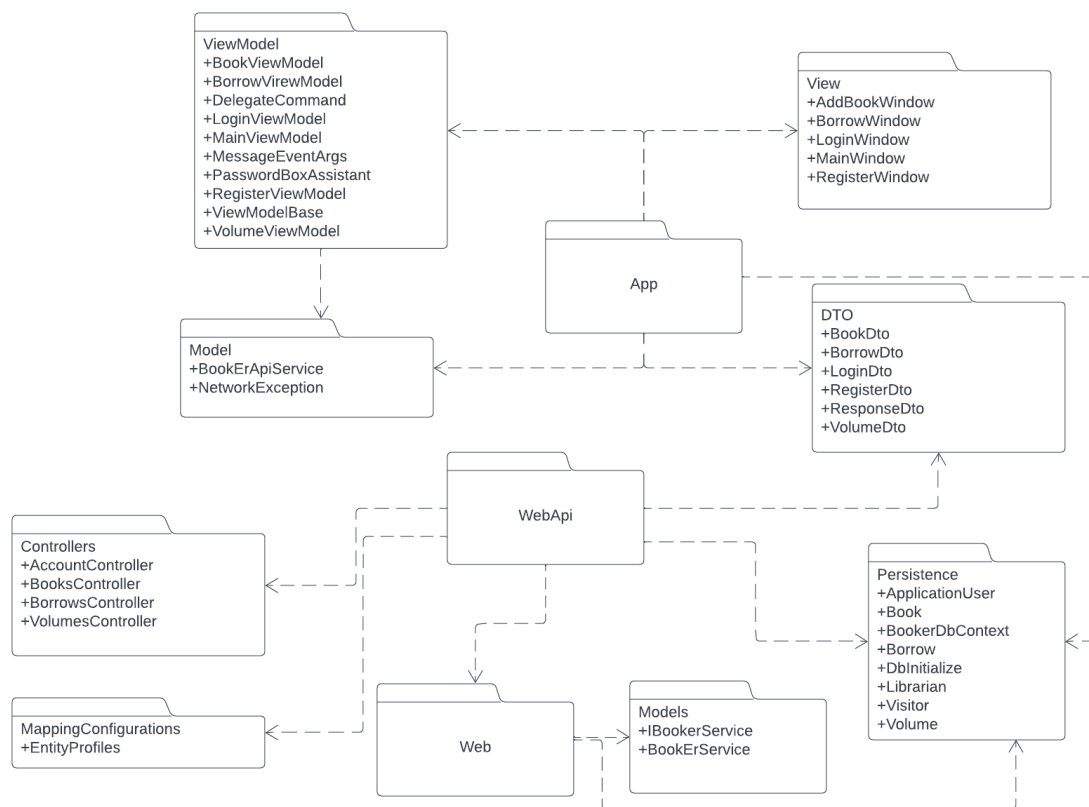
Feladat Elemzése:

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 részfaladatban 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.

BookDto
+ Id : int + Title : String + Author : String + Year : int + ISBN : String + Image: Byte[]

LoginDto
+ UserName : String + Password : String

RegisterDto
+ Name : String + UserName : String + Email : String + PhoneNumber : String + Password : String

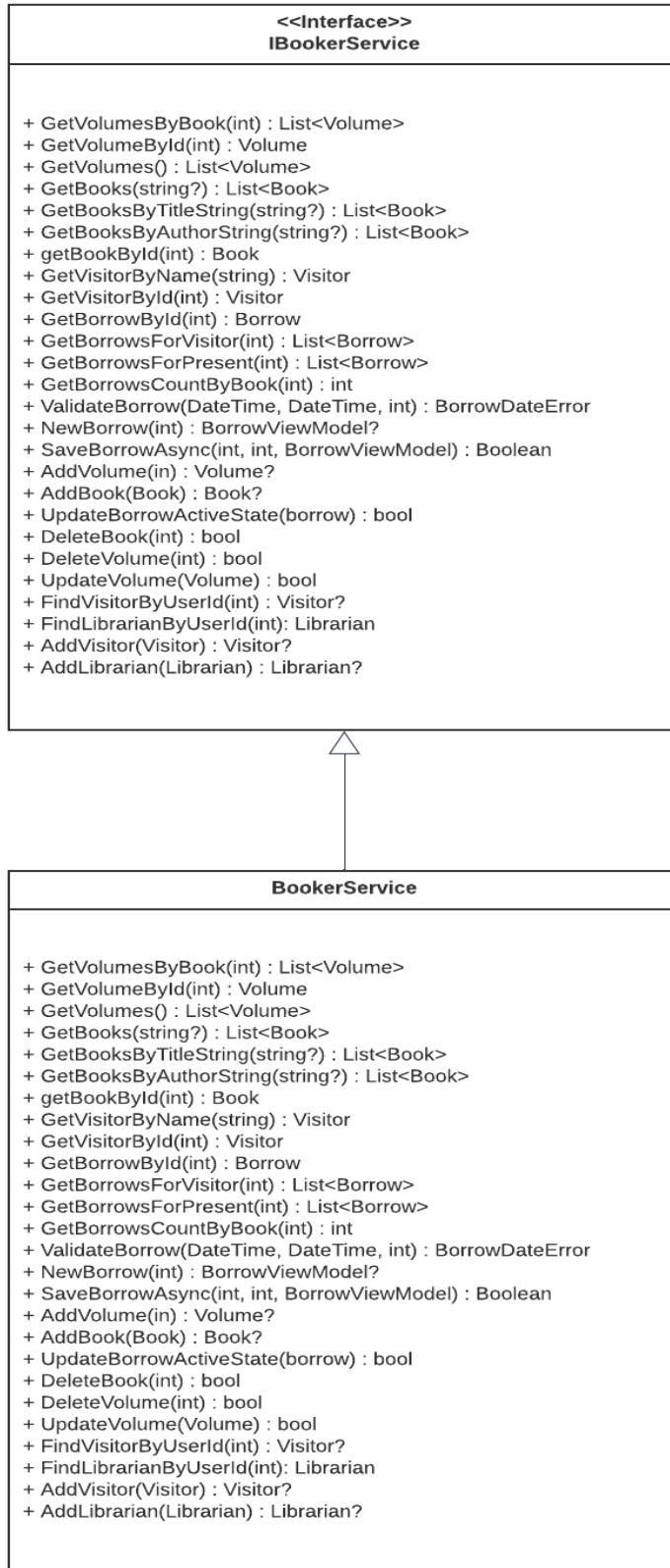
BorrowDto
+ Id : int + Volumeld : int + VisitorId : int + StartDay : DateTime + EndDay : DateTime + IsActive : Boolean + UserName : String

ResponseDto
+ Success : Boolean + ErrorMessage : String

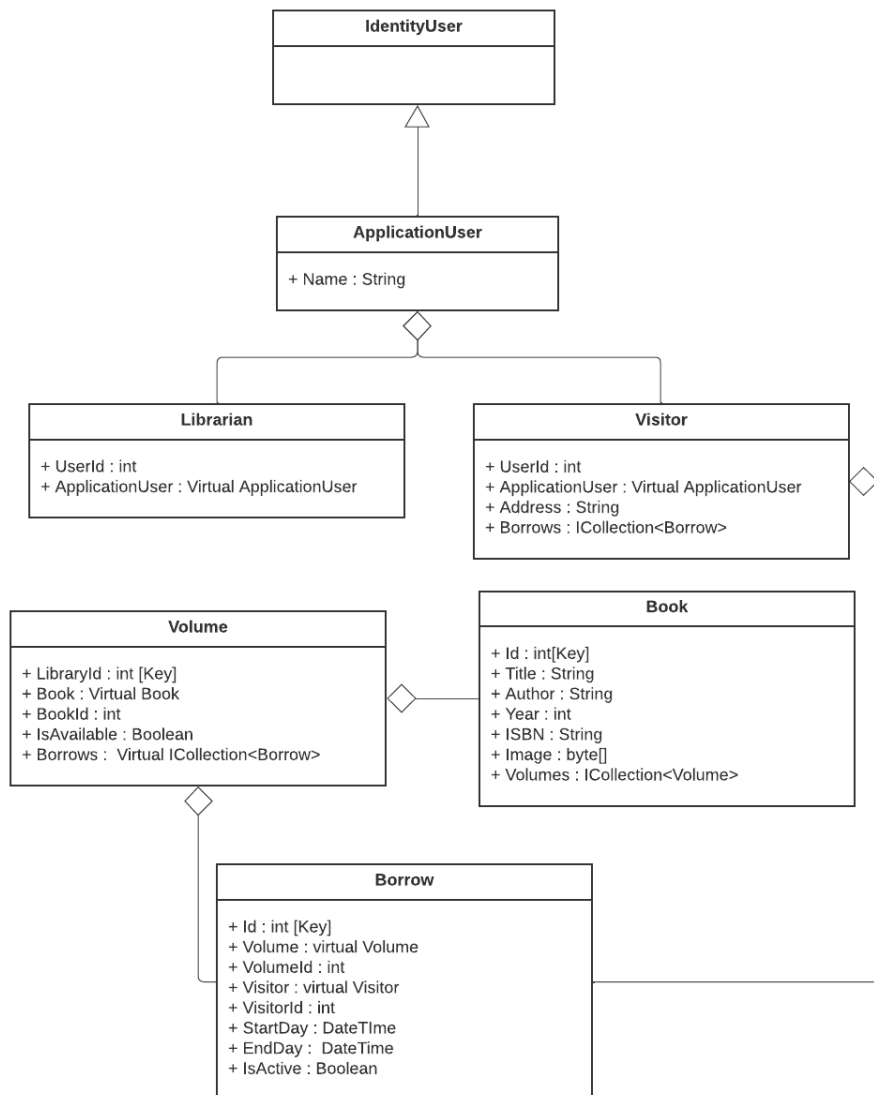
VolumeDto
+ LibraryId : int + BookId : int + IsAvailable : Boolean + CurrentUser : String + CurrentDeadline : DateTime + NextBorrow : DateTime

Services

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



Adatbázis



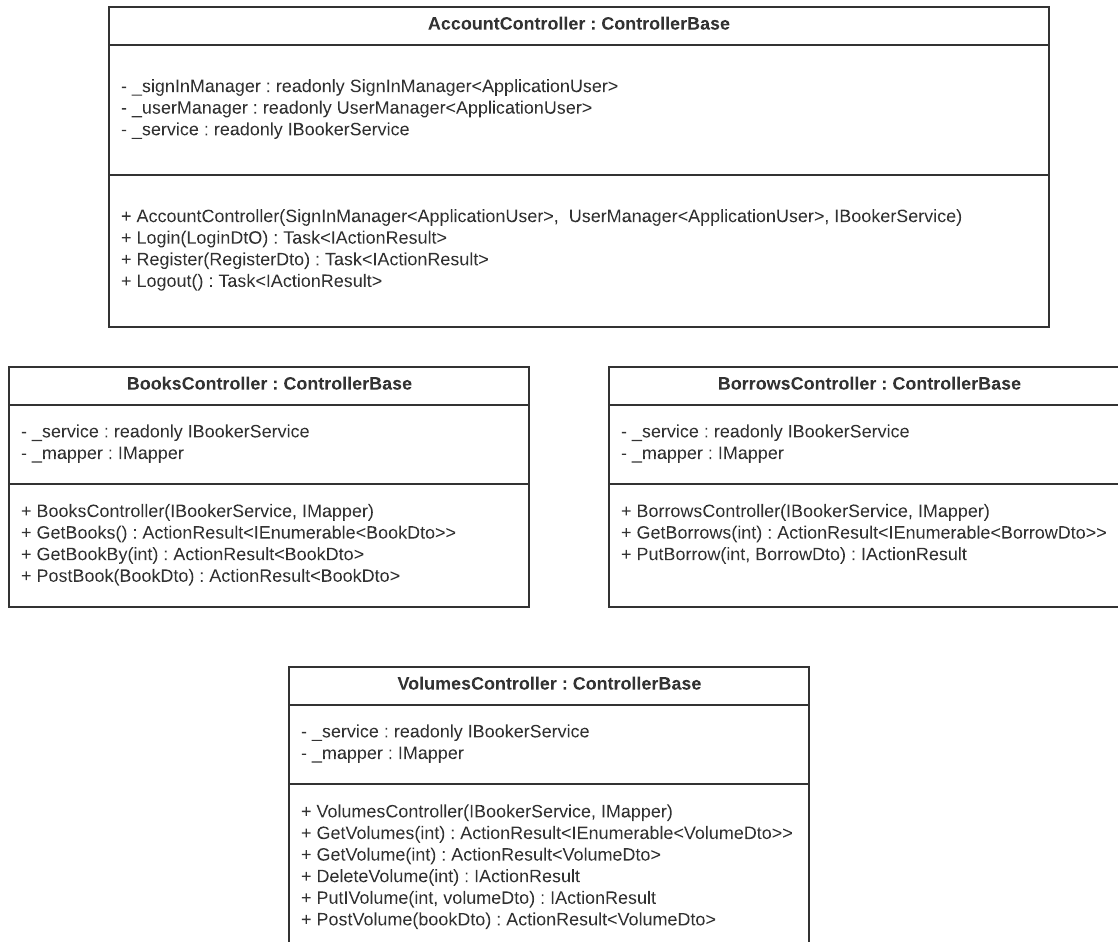
WebApi projekt (BookEr.WebApi)

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, DELETE). 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 controllerben. 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 Controller 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.

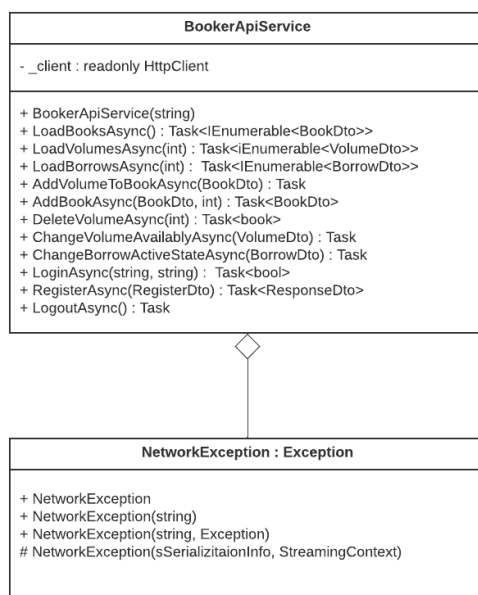
Az asztali alkalmazás nézet-modell-nézetmodell (model-view-viewmodel MVVM) architektúrában van megvalósítva. A modell az üzleti logikát (ez esetben a webapival való kommunikációt) valósítja meg, míg a nézet az ablakok és az azokban található elemek megjelenítéséért felel. A nézetmodell a megjelenítésért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

Az MVVM architektúra segít elkülöníteni az alkalmazás grafikus felületét (nézet) és a háttérben 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.

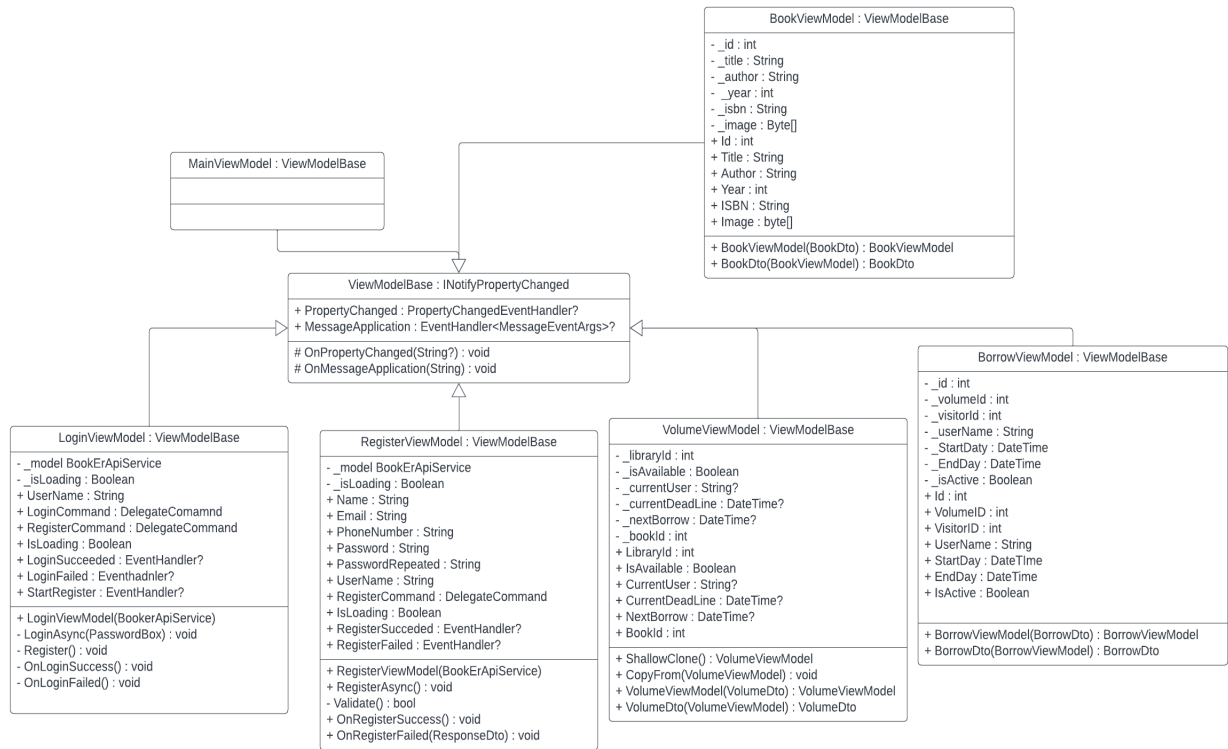
Ha a kommunikáció során kivétel keletkezik, akkor egy általunk definiált kivételt dobunk (Network Exception), amelynek az üzenetében továbbítjuk a hibakódot.



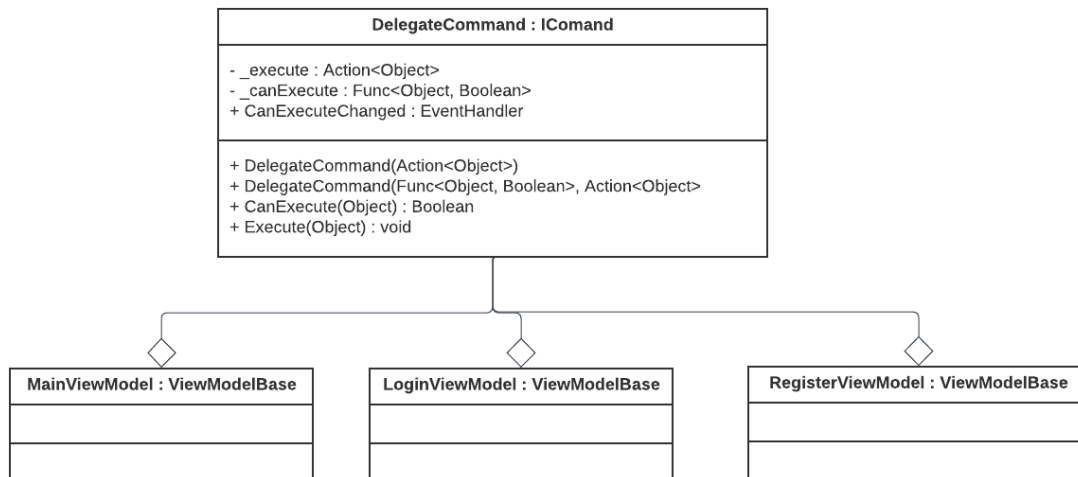
Nézetmodell réteg

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

MainViewModel : ViewModelBase
<div><div><div>- _service : readonly BookErApiService</div><div>- _books : ObservableCollection<BookViewModel>?</div><div>- _volumes : ObservableCollection<VolumeViewModel>?</div><div>- _borrows : ObservableCollection<BorrowViewModel>?</div><div>- _selectedVolume : VolumeViewModel?</div><div>- _selectedBook : BookViewModel?</div><div>- _selectedBorrow : BorrowViewModel?</div><div>- _newBook : BookViewModel?</div><div>- _isItAvailable : String</div><div>- _selectedBookTitle : String?</div><div>- _newBookVolumeCount : uint</div><div>+ IsItAvailable : String</div><div>+ Books : ObservableCollection<BookViewModel>?</div><div>+ Volumes : ObservableCollection<VolumeViewModel>?</div><div>+ Borrows : ObservableCollection<BorrowViewModel>?</div><div>+ SelectedVolume : VolumeViewModel?</div><div>+ SelectedBook : BookViewModel?</div><div>+ SelectedBorrow : BorrowViewModel?</div><div>+ NewBookVolumeCount : uint</div><div>+ NewBook : BookViewModel?</div><div>+ SelectedBookTitle : String?</div><div>+ RefreshBooksCommand : DelegateCommand</div><div>+ SelectCommand : DelegateCommand</div><div>+ VolumeSelectCommand : DelegateCommand</div><div>+ LogoutCommand : DelegateCommand</div><div>+ DeleteColumeCommand : DelegateCommand</div><div>+ ChangeAvailableStateCommand : DelegateCommand</div><div>+ AddVolumeCommand : DelegateCommand</div><div>+ AddBookCommand : DelegateCommand</div><div>+ CancelAddBookCommand : DelegateCommand</div><div>+ CangelImageCommand : DelegateCommand</div><div>+ SaveNewBookCommand : DelegateCommand</div><div>+ ShowBorrowsCommand : DelegateCommand</div><div>+ LogoutSucceded : EventHandler?</div><div>+ StartingAddBook : EventHandler?</div><div>+ FinishingAddBook : EventHandler?</div><div>+ StartingImageChange : EventHandler?</div><div>+ ShowBorrows : EventHandler?</div></div></div> <div><div>+ MainViewModel(BookerApiService)</div><div>+ LoadBooksAsync() : void</div><div>+ LoadVolumesAsync(BookViewModel) : void</div><div>+ ShowBorrowsAsync(VolumeViewModel) : void</div><div>+ LogoutAsync() : void</div><div>+ AddBookStart() : void</div><div>+ CancelAddBook() : void</div><div>+ DeleteVolume(VolumeViewModel) : void</div><div>+ ChangeActiveState(BorrowViewModel) : void</div><div>+ ChangeBorrowedButtonDisplayAsync(VolumeViewModel) : void</div><div>+ AddVolumeToBook(BookViewModel) : void</div><div>+ SaveNewBook() : void</div></div>



DelegateCommand kötése



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.

App : Application
<ul style="list-style-type: none">- _service : BookErApiService- _mainViewModel : MainViewModel- _loginViewModel : LoginViewModel- _registerViewModel : RegisterViewModel- _mainView : MainWindow- _loginView : LoginWindow- _addBookView : AddBookWindow- _borrowView : BorrowWindow- _registerView : RegisterWindow
<ul style="list-style-type: none">+ App()- App_Startup(object, StartupEventArgs) : void- ViewModel_MessageApplication(Object, MessageEventArgs) : void- LoginView_Closed(object, EventArgs) : void- ViewModel_LoginSucceeded(object, EventArgs) : void- ViewModel_RegisterSucceeded(object, EventArgs) : void- ViewModel_StartRegister(object, EventArgs) : void- ViewModel_LoginFailed(object, EventArgs) : void- ViewModel_LogoutSucceeded(object, EventArgs) : void- ViewModel_StartingAddBook(object, EventArgs) : void- ViewModel_FinishingAddBook(object, EventArgs) : void- ViewModel_StartingImageChange(object, EventArgs) : void- ViewModel_ShowBorrowWindow(object, EventArgs) : void

Tesztelés

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

BooksController:

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

GetBookByIdTest: Három különböző id-val lekérünk egy-egy könyvet, majd ellenőrizzük, hogy az eredmény id-ja megegyezik-e az általunk 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 számosságát.

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 számosságát.

GetVolumesInvalidTest: Nemlétező könyvhöz kérünk le kölcsönzéseket.

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

DeleteVolumeTest: Kitörölünk egy kötetet, majd megpróbáljuk lekérni újra az adatbázisból. Ellenőrizzük, 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.