2.Beadandó feladat dokumentáció

Készítette: Kovács Máté

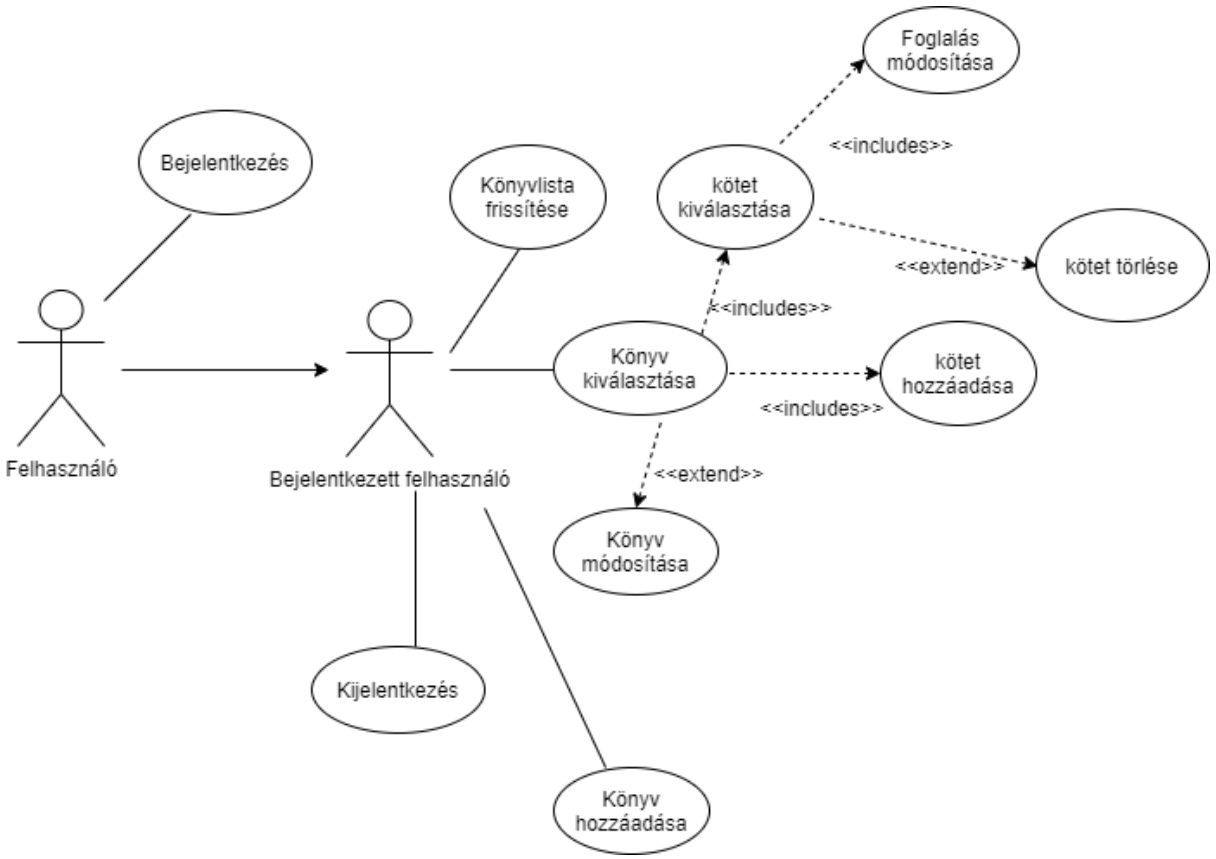
E-mail: hstgu5@inf.elte.hu

Feladat:

2. részfeladat: 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.

Elemzés:

• Az második részfeladatot a .NET Core keretrendszerben készítjük el, az adatbázis kezelését az Entity Framework Core segítségével végezzük.
• A felhasználókezelést és autentikációt a keretrendszer Identity kezelő részével végezzük.
• Az api szolgáltatás eléréséhez feltétel a bejelentkezés, ez után a szolgáltatáson segítségével data transfer objektumokon keresztül tudunk az adatbázisból könyveket feltölteni, köteteket hozzáadni/törölni, foglalásokat aktiválni illetve deaktiválni.
• Az asztali alkalmazást MVVM architektúrában, wpf grafikus felülettel valósítjuk meg, ebben tudja a felhasználó bejelentkezést követően az api szolgáltatáson keresztül a fentebb felsorolt funkciókat kezelni.

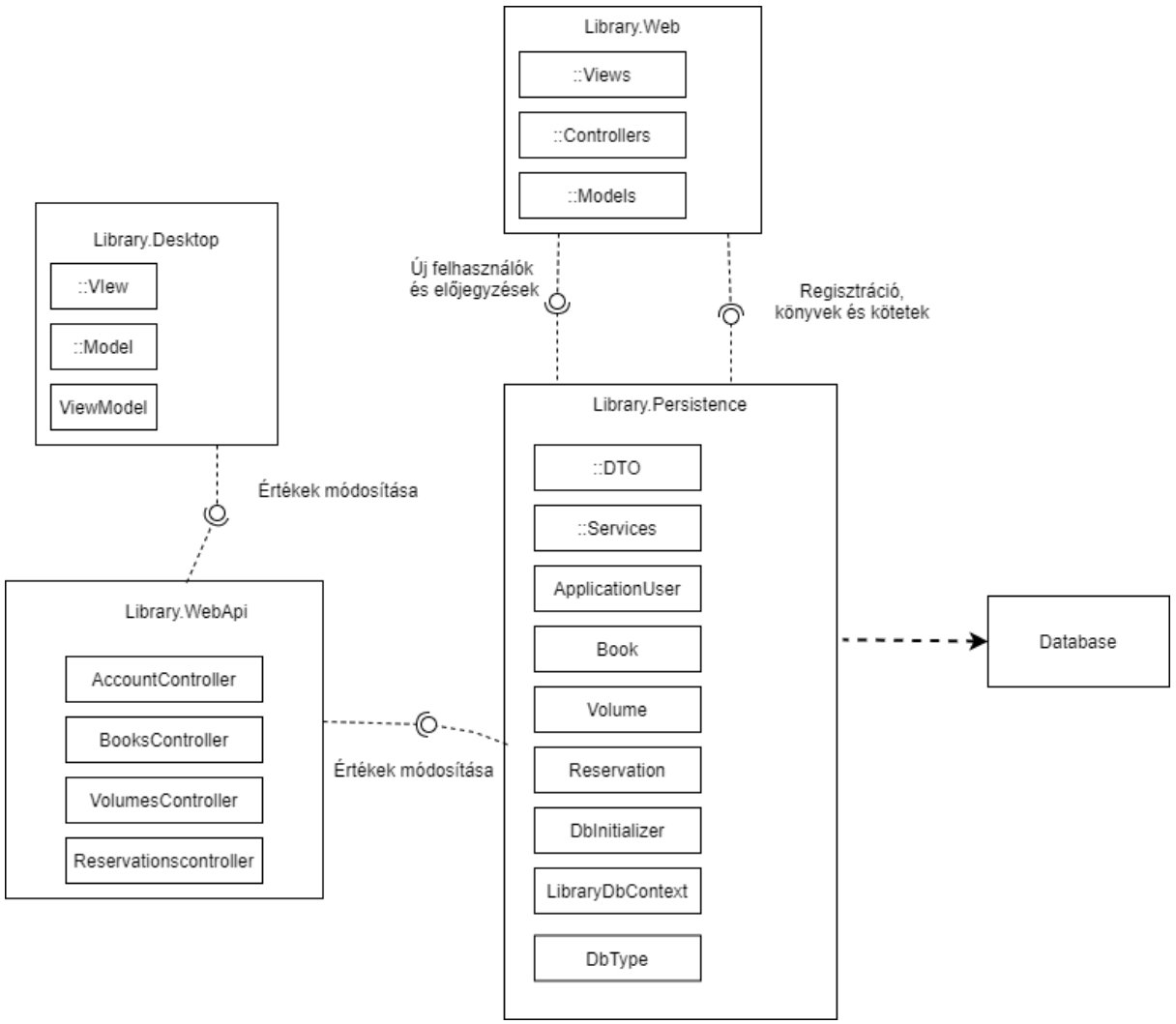


1.ábra: felhasználói esetek diagramja

Tervezés

Programszerkezet:

A korábbi egyprojektes MVC felépítésű programot kibővítve több projektet hozunk létre, a korábbi weboldal nézeteit és kontrollereit a Library.Web, az entitás, az adatbázis inicializátor és kontextus osztályokat, azonkívül a DTO objektumokat a Library.Persistence,az api szolgáltatást a Library.WebApi, az asztali alkalmazást a Library.Desktop projektekbe helyezzük ki. Ezen kívül az api teszteléséhez még létrehoztuk az xUnit keretrendszert használó tesztelő projektet.

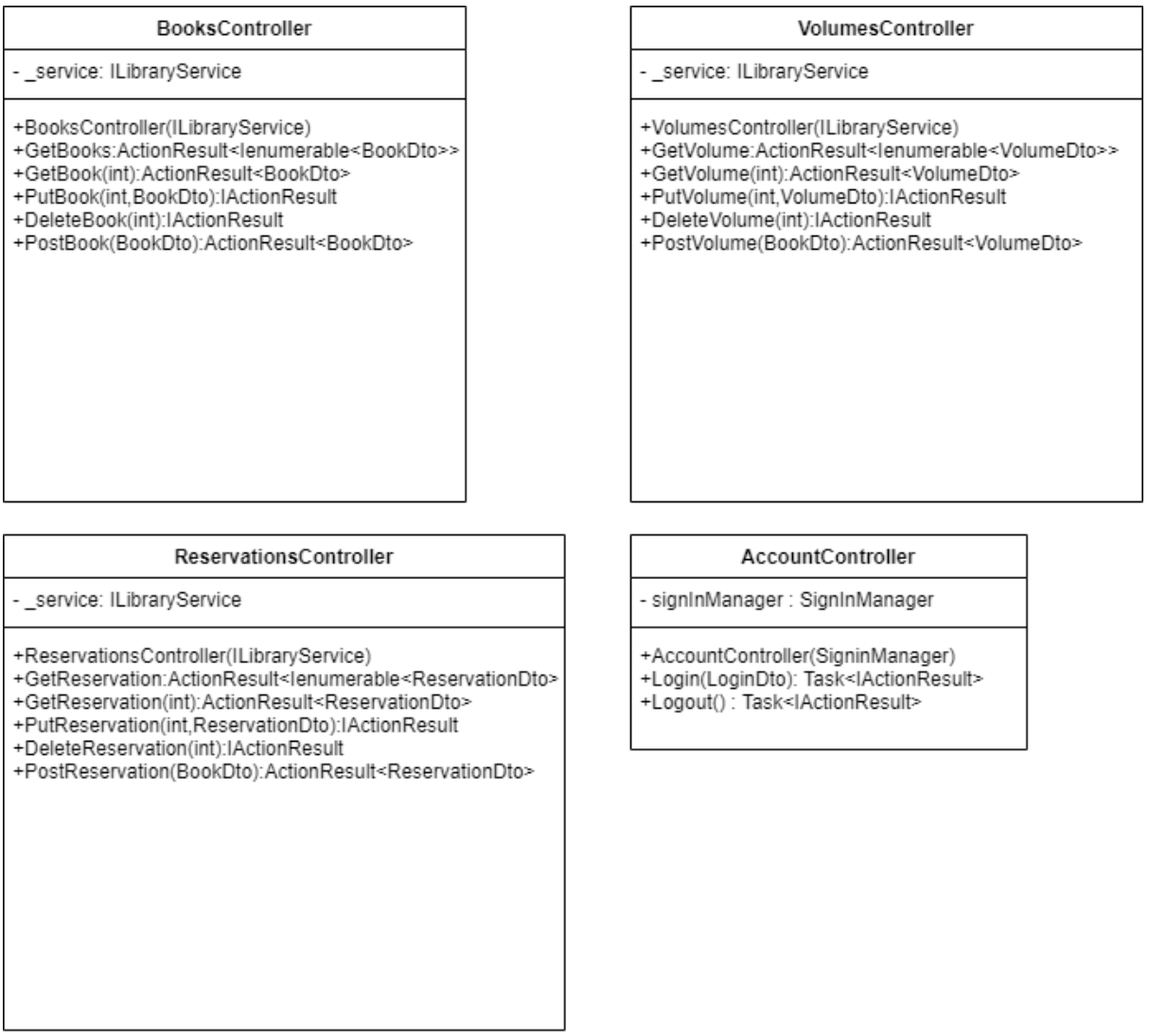


2.ábra: a program komponensdiagramja.

Web API service

Az api szolgáltatás feladata, hogy az asztali alkalmazás felé adatbázishozzáférést biztosítson. Az AccountController Login és Logout metódusai a felhasználó ki- és bejelentkeztetését végzik, illetve bejelentkezett felhasználó számára lehetőség van a **BooksController**, **VolumesController**, és **ReservationsController** segítségével az adott adatok lekérésére.

A hozzáférést a szolgáltatás nem közvetlenül a tárolt entitásokkal, hanem az ezekből képzett Data Transfer Object-ekkel nyújtja, ezek a Library.Persistence-ben vannak tárolva. A kérések teljesítése közben akadó problémákról hibákkal jelez vissza a szolgáltatás.



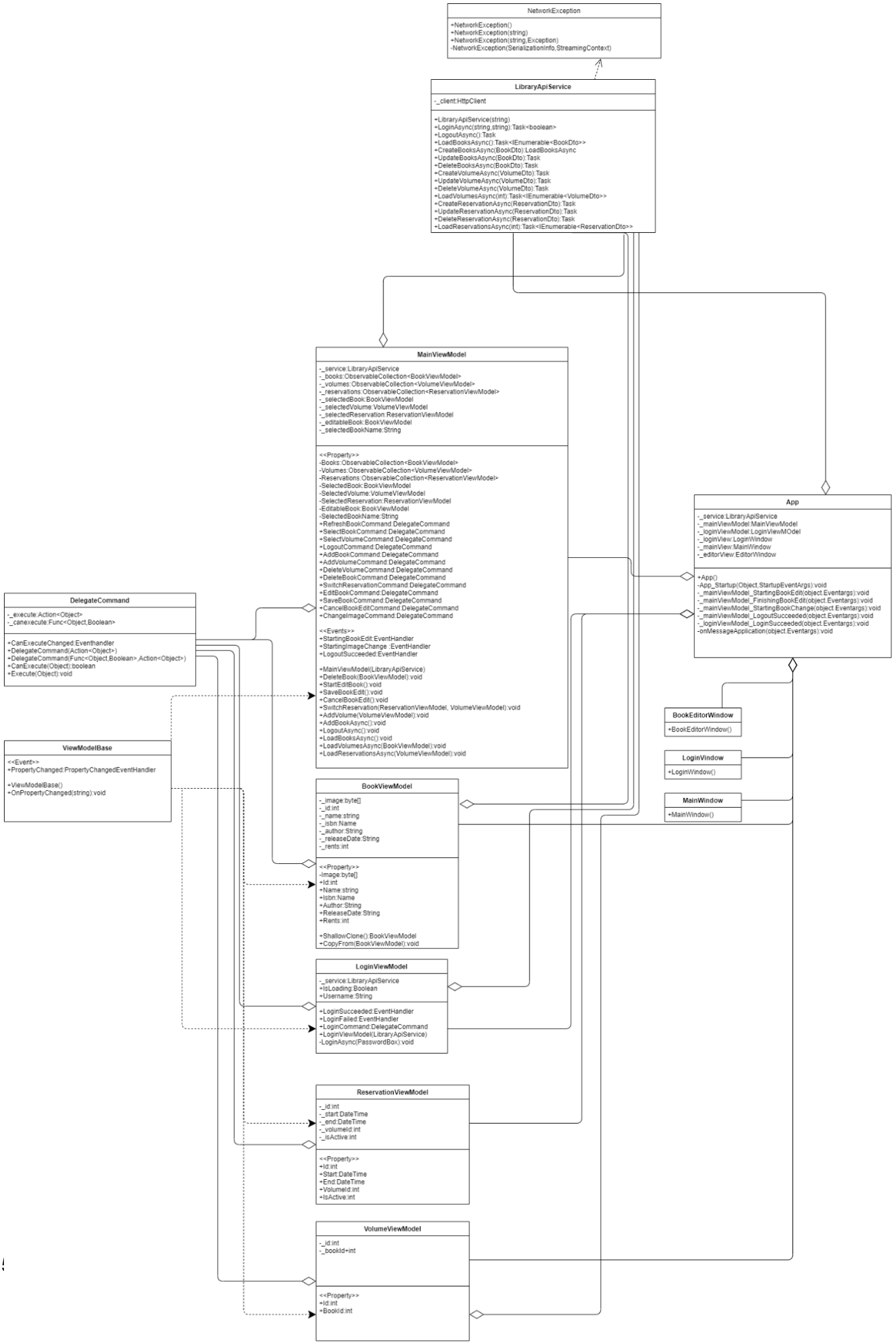
A Web API osztálydiagramja

Asztali alkalmazás

Az asztali alkalmazást MVVM architektúrában, WPF grafikus felülettel valósítjuk meg.

A 3 nézet a bejelentkezés, a könyvtári adatok listázása, és módosítása, egy könyv szerkesztése feladatköröket oldják meg.

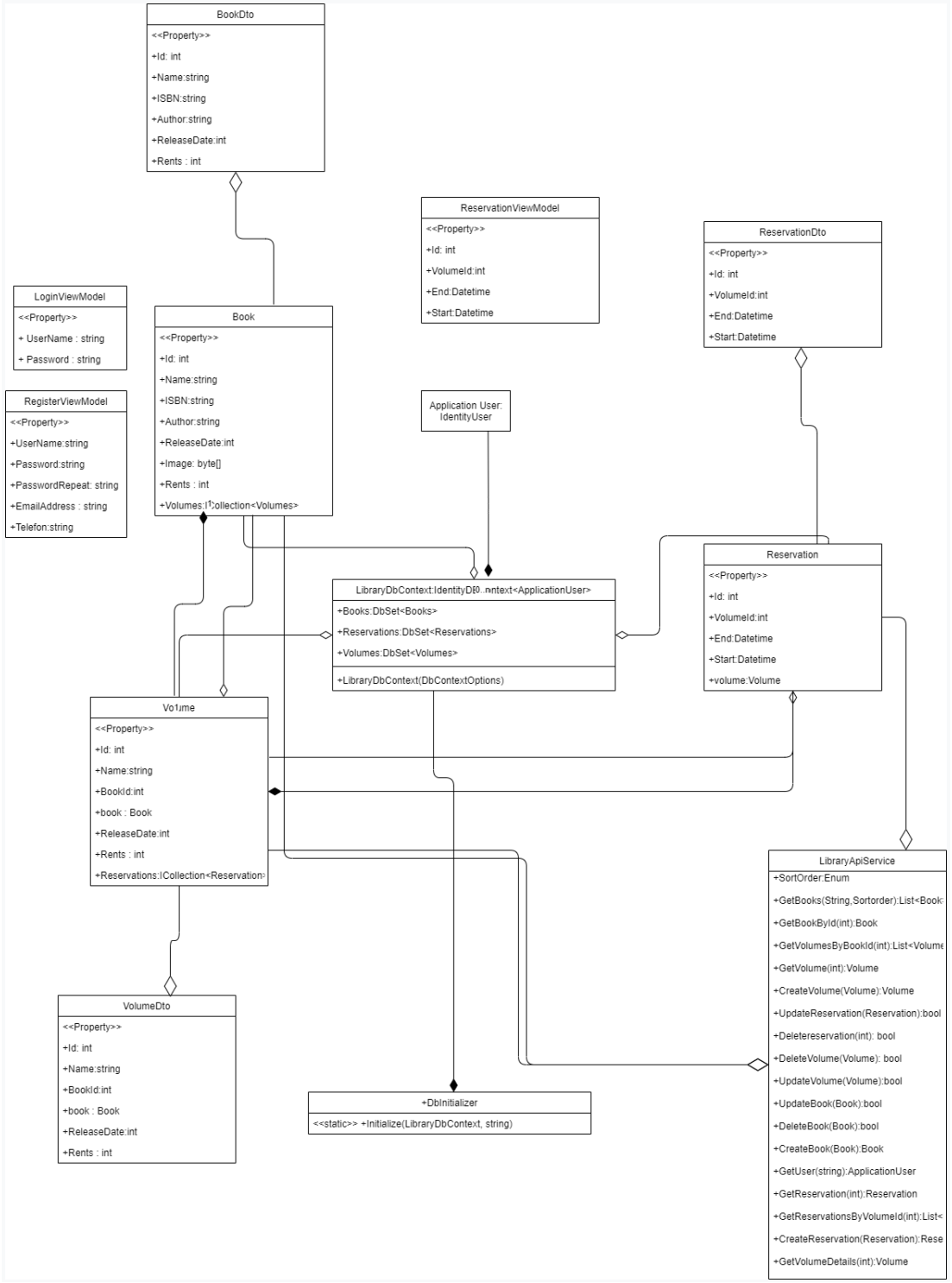
A nézetmodellek felelősek a nézetek és a modellek közti adatcseréért, illetve az App osztály végzi a megfelelő nézet – nézetmodell párok közti váltogatást, illetve a megfelelő függőségek átadását.


NetworkException
+NetworkException()
+NetworkException(string)
+NetworkException(string,Exception)
-NetworkException(SerializationInfo,StreamingContext)
LibraryApiService
-_client:HttpClient
+LibraryApiService(string)
+LoginAsync(string,string):Task<boolean>
+LogoutAsync():Task
+LoadBooksAsync():Task<IEnumerable<BookDto>>
+CreateBooksAsync(BookDto):LoadBooksAsync
+UpdateBooksAsync(BookDto):Task
+DeleteBooksAsync(BookDto):Task
+CreateVolumeAsync(VolumeDto):Task
+UpdateVolumeAsync(VolumeDto):Task
+DeleteVolumeAsync(VolumeDto):Task
+LoadVolumesAsync(int):Task<IEnumerable<VolumeDto>>
+CreateReservationAsync(ReservationDto):Task
+UpdateReservationAsync(ReservationDto):Task
+DeleteReservationAsync(ReservationDto):Task
+LoadReservationsAsync(int):Task<IEnumerable<ReservationDto>>
MainViewModel
-_service:LibraryApiService
-_books:ObservableCollection<BookViewModel>
-_volumes:ObservableCollection<VolumeViewModel>
-_reservations:ObservableCollection<ReservationViewModel>
-_selectedBook:BookViewModel
-_selectedVolume:VolumeVIewModel
-_selectedReservation:ReservationViewModel
-_editableBook:BookViewModel
-_selectedBookName:String
<<Property>>
-Books:ObservableCollection<BookViewModel>
-Volumes:ObservableCollection<VolumeViewModel>
-Reservations:ObservableCollection<ReservationViewModel>
-SelectedBook:BookViewModel
-SelectedVolume:VolumeVIewModel
-SelectedReservation:ReservationViewModel
-EditableBook:BookVIewModel
-SelectedBookName:String
+RefreshBookCommand:DelegateCommand
+SelectBookCommand:DelegateCommand
+SelectVolumeCommand:DelegateCommand
+LogoutCommand:DelegateCommand
+AddBookCommand:DelegateCommand
+AddVolumeCommand:DelegateCommand
+DeleteVolumeCommand:DelegateCommand
+DeleteBookCommand:DelegateCommand
+SwitchReservationCommand:DelegateCommand
+EditBookCommand:DelegateCommand
+SaveBookCommand:DelegateCommand
+CancelBookEditCommand:DelegateCommand
+ChangeImageCommand:DelegateCommand
<<Events>>
+StartingBookEdit:EventHandler
+StartingImageChange :EventHandler
+LogoutSucceeded:EventHandler
+MainViewModel(LibraryApiService)
+DeleteBook(BookViewModel):void
+StartEditBook():void
+SaveBookEdit():void
+CancelBookEdit():void
+SwitchReservation(ReservationViewModel, VolumeViewModel):void
+AddVolume(VolumeViewModel):void
+AddBookAsync():void
+LogoutAsync():void
+LoadBooksAsync():void
+LoadVolumesAsync(BookViewModel):void
+LoadReservationsAsync(VolumeViewModel):void
App
-_service:LibraryApiService
-_mainViewModel:MainViewModel
-_loginViewModel:LoginViewMOdel
-_loginView:LoginWindow
-_mainView:MainWindow
-_editorView:EditorWindow
+App()
-App_Startup(Object,StartupEventArgs):void
-_mainViewModel_StartingBookEdit(object,Eventargs):void
-_mainViewModel_FinishingBookEdit(object,Eventargs):void
-_mainViewModel_StartingBookChange(object,Eventargs):void
-_mainViewModel_LogoutSucceeded(object,Eventargs):void
-_loginViewModel_LoginSucceeded(object,Eventargs):void
-onMessageApplication(object,Eventargs):void
DelegateCommand
-_execute:Action<Object>
-_canexecute:Func<Object,Boolean>
+CanExecuteChanged:Eventhandler
+DelegateCommand(Action<Object>)
+DelegateCommand(Func<Object,Boolean>,Action<Object>)
+CanExecute(Object):boolean
+Execute(Object):void
ViewModelBase
<<Event>>
+PropertyChanged:PropertyChangedEventHandler
+ViewModelBase()
+OnPropertyChanged(string):void
BookViewModel
-_image:byte[]
-_id:int
-_name:string
-_isbn:Name
-_author:String
-_releaseDate:String
-_rents:int
<<Property>>
-Image:byte[]
+Id:int
+Name:string
+Isbn:Name
+Author:String
+ReleaseDate:String
+Rents:int
+ShallowClone():BookViewModel
+CopyFrom(BookViewModel):void
LoginViewModel
-_service:LibraryApiService
+IsLoading:Boolean
+Username:String
+LoginSucceeded:EventHandler
+LoginFailed:EventHandler
+LoginCommand:DelegateCommand
+LoginViewModel(LibraryApiService)
-LoginAsync(PasswordBox):void
ReservationViewModel
-_id:int
-_start:DateTime
-_end:DateTime
-_volumeId:int
-_isActive:int
<<Property>>
+Id:int
+Start:DateTime
+End:DateTime
+VolumeId:int
+IsActive:int
VolumeViewModel
-_id:int
-_bookId+int
<<Property>>
+Id:int
+BookId:int
BookEditorWindow
+BookEditorWindow()
LoginVindow
+LoginWindow()
MainWindow
+MainWindow()

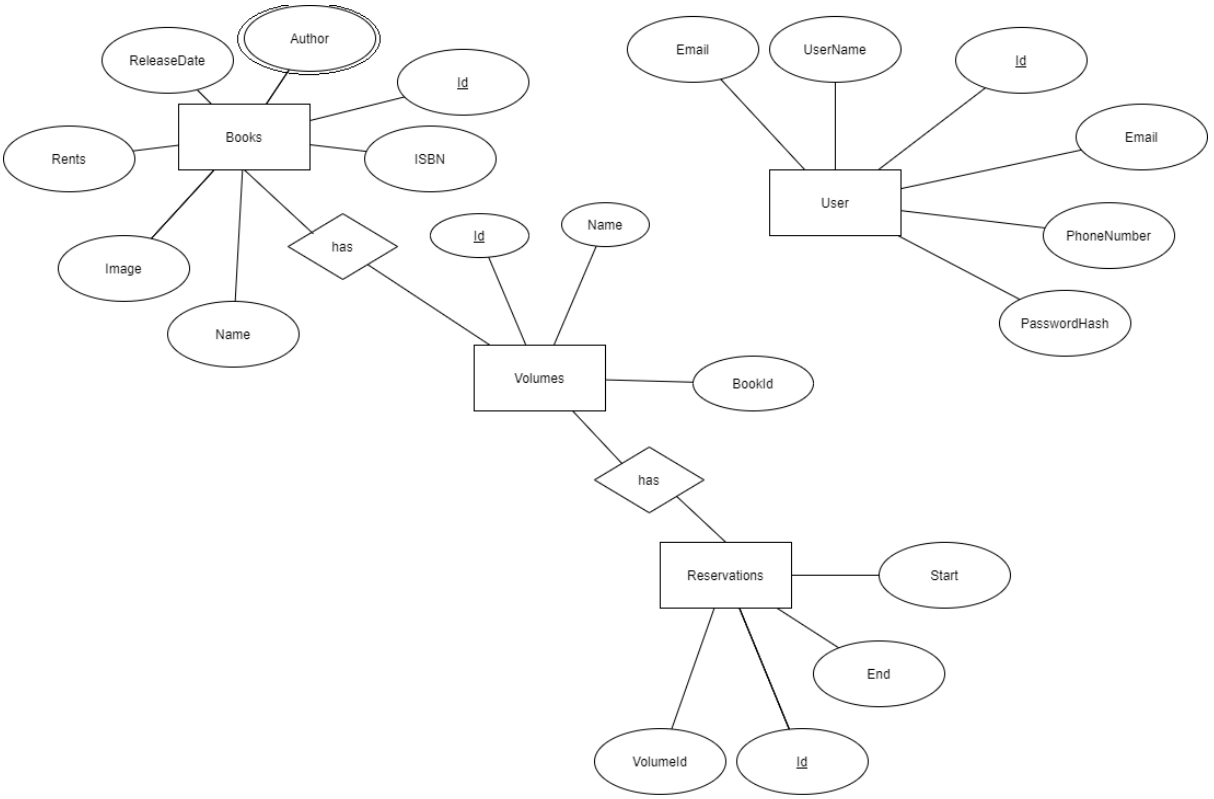
Asztali alkalmazás osztálydiagramja.

Model

A model projektben található az adatbázis code-first megközelítésű leírása, az entitás osztályokkal, és az ezeket tároló kontextussal. Ezen kívül tartalmaz egy inicializáló osztályt is, ami létrehozza a szükséges adatbázist.



A model osztálydiagramja.



3.Ábra: az adatbázis EK-diagramja

Az adatbázis

Az Entity Framework Core segítségével kezelt adatbázisban tároljuk a könyveket és azok köteteit, a kötetekre leadott foglalásokat, és a regisztrált felhasználókat.

API felület

A szolgáltatás végpontjai a futtató tartományon belül a következő útvonalakon helyezkednek el, funkcióik a következők:

- Az **api/account/login/** végponton egy Post küldés törzsében felhasználónévvel és jelszóval bejelentkezhetünk.
- Az **api/account/logout/** végponton a bejelentkezett felhasználók kijelentkezhetnek .
- Az **api/Books/** végponton megtalálható a könyvtári katalógus.
- Az **Api/Volumes/Book/Id** végponton a kötetek érhetőek el.
- Az **Api/Reservations/Volumes/Id** végponton pedig a kötetekhez tartozó foglalások találhatók.

Tesztelés:

Az api szolgáltatás funkcionalitását xUnit egységtesztekkel ellenőrizzük a ServiceTest osztályban. Ebben a konstruktorban mindig új, memória-beli adatbázist hozunk létre, illetve feltöltjük tesztadatokkal. Ezen kívül az osztály megvalósítja az IDisposable interfészt, így az egyes tesztesetek között töröljük az adatbázist, és nem hatnak ki egymásra az egységtesztek.A tesztek a hozzájuk tartozó, azonos nevű osztályokat tesztelik, ezek pedig a BooksControllerTest, VolumesControllerTest, és a ReservationsControllerTest osztályok.A tesztelésre létrehozott osztályok elemzése:

- **TestDbInitializer**: a tesztek számára hoz létre egy ideiglenes adatbázist, tartalma majdnem teljesen megegyezik a Library.Persistence osztályba tartozó DbInitializer-el.
- **BooksController**
    - GetbooksTest: Lekérdezzük a könyveket az adatbázisból, majd ellenőrizzük, hogy a könyvek száma megegyezik ez az adatbázisban tartalmazott könyvek számával
    - GetBookByIdTest: Lekérdezünk egy könyvet a paraméterben megadott Id segítségével, majd ellenőrizzük, hogy a lekérdezett könyv ID-je megegyezik e a paraméterrel.
    - GetInvalidBookTest: Lekérdezünk egy könyvet egy olyan ID-n amihez nem tartozik könyv, és ellenőrizzük hogy megfelelő hibaüzenetet kapunk-e vissza.
    - PostBookTest:Létrehozunk egy új könyvet, majd feltöltjük az adatbázisba.Ez után teszteljük, hogy az adatbázisban tartalmazott könyvek száma megnőtt-e.
- **VolumesController**
    - GetVolumesTest: Lekérdezünk egy könyvhez tartozó köteteket az adatbázisból, majd ellenőrizzük, hogy a kötetek száma megegyezik ez az adatbázisban tartalmazott kötetek számával.
    - GetVolumeByIdTest: Lekérdezünk egy kötetet a paraméterben megadott Id segítségével, majd ellenőrizzük, hogy a lekérdezett kötet ID-je megegyezik e a paraméterrel.
    - GetInvalidVolumeTest: Lekérdezünk egy kötetet egy olyan ID-n amihez nem tartozik kötet, és ellenőrizzük hogy megfelelő hibaüzenetet kapunk-e vissza.
    - PostVolumeTest:Létrehozunk egy új kötetet, majd feltöltjük az adatbázisba.Ez után teszteljük, hogy az adatbázisban tartalmazott kötetek száma megnőtt-e.
- **ReservationsController**
    - GetReservationsTest: Lekérdezünk egy kötethez tartozó foglalásokat az adatbázisból, majd ellenőrizzük, hogy a foglalások száma megegyezik ez az adatbázisban tartalmazott foglalások számával.

- GetReservationByIdTest: Lekérdezünk egy foglalást a paraméterben megadott Id segítségével, majd ellenőrizzük, hogy a lekérdezett foglalás ID-je megegyezik e a paraméterrel.
- GetInvalidBookTest: Lekérdezünk egy foglalást egy olyan ID-n amihez nem tartozik foglalás, és ellenőrizzük hogy megfelelő hibaüzenetet kapunk-e vissza.