2. Beadandó feladat dokumentációja

Készítette:

Név	Peskó Márton
Azonosító	YRQHGX
E-mail cím	peskomarton@hotmail.com

Feladat leírása:

(7.) Könyvtári kölcsönző

Készítsük el egy könyvtár online kölcsönzői és nyilvántartó rendszerét, amellyel a látogatók kölcsönzését, előjegyzését, valamint a könyvtárosok adminisztratív munkáját tudjuk támogatni.

(2.) részfeladat

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

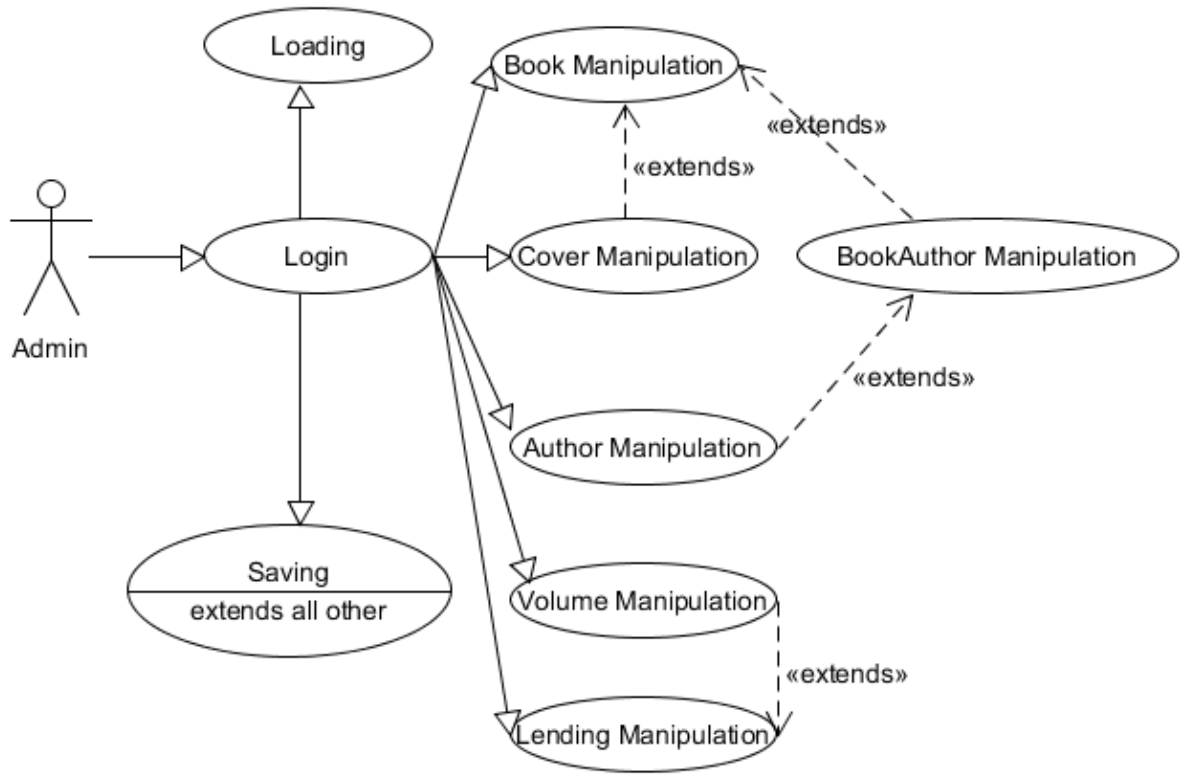
- A könyvtáros bejelentkezhet (felhasználónév és jelszó megadásával) a programba, illetve kijelentkezhet; a további funkcionalitások csak bejelentkezett állapotban elérhetőek.
- Az alkalmazás listázza a könyveket, valamint a hozzájuk tartozó köteteket. Lehetőség van új könyv, illetve kötet rögzítésére.
- 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.

Az adatbázis az alábbi adatokat tárolja:

- könyvek (cím, szerző, kiadás éve, ISBN szám, borítókép);
- kötetek (könyv, könyvtári azonosító);
- kölcsönzések (kötet, látogató, kezdő nap, befejező nap, aktív-e)
- látogatók (név, cím, telefonszám, azonosító, jelszó);
- könyvtárosok (név, azonosító, jelszó).

Elemzés:

A megoldást [.NET] WPF MVVM keretrendszerrel és egy [.NET Core 2] WEB API-val meg lehet valósítani.



Nézetek:

Kelleni fog egy a bejelentkezéshez. Itt egy egyszerű felhasználónév jelszó adatokat bekérő mezőket tartalmazó ablakról lesz szó.

Valamint szükség lesz még egyre, amelyen megjelennek az adatok és a hozzájuk kapcsolódó manipulációkat lehetővé tevő gombok és menü pontok. Ezt az ablakot egy menüvel és egy címkés konténerrel fogom kitölteni. Két címke lesz egy a könyveknek és a hozzájuk kapcsolódó adatoknak, valamint egyet a kölcsönzésekhez.

Kellene fog adatbevitelhez két újabb ablak, mivel a könyvek és a szerzők adatbevitele alig korlátozott. Ezeken fog megtörténni az adat felvétel az új és az szerkesztet könyvekhez és szerzőkhöz.

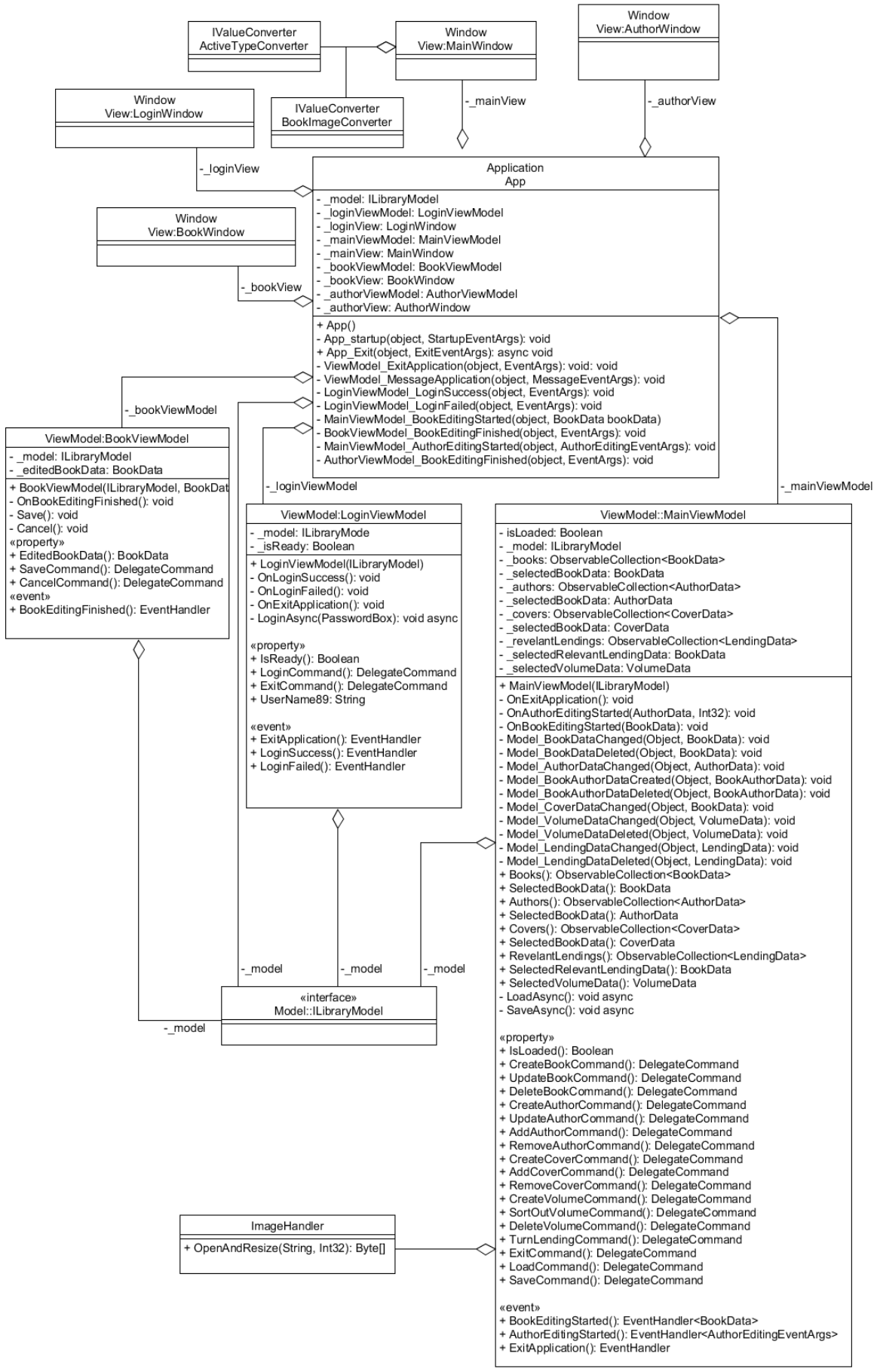
Kontroller:

Lesz egy osztály [App.xaml.cs], amely elinduláskor elindítja a bejelentkezés nézetét. Ez az osztály fogja fogadni és összekötni a programban kiváltott eseményeket, valamint nézeteket elindítani és bezárni.

Nézet modellek:

Lesz egy bázis nézet modellünk, hogy a nézet modellink által kiváltott nézetben megjelenő adatok változását automatikusan jelenítse meg. Ebből a nézet modellből származtatjuk a többit. Minden nézet kap egy saját nézet modellt, melyek majd kezelik a nézeten kiváltott eseményeket.

A bejelentkezés ablakhoz tartotó nézet modell ellenőrizteti az adatokat a WEB API-val és annak, válaszának függvényében küld üzenetet a kontrollernek. Sikertelene esetben, a felhasználónak van további lehetősége próbálkozni. Sikere esetén a kontrollere megnyitja a fő ablakot és bezárja a bejelentkezést lehetővé tevő ablakot.


IValueConverter
ActiveTypeConverter
Window
View:MainWindow
Window
View:AuthorWindow
Window
View:LoginWindow
IValueConverter
BookImageConverter
-_mainView
-_authorView
-_loginView
Application
App
- _model: ILibraryModel
- _loginViewModel: LoginViewModel
- _loginView: LoginWindow
- _mainViewModel: MainViewModel
- _mainView: MainWindow
- _bookViewModel: BookViewModel
- _bookView: BookWindow
- _authorViewModel: AuthorViewModel
- _authorView: AuthorWindow
+ App()
- App_startup(object, StartupEventArgs): void
+ App_Exit(object, ExitEventArgs): async void
- ViewModel_ExitApplication(object, EventArgs): void: void
- ViewModel_MessageApplication(object, MessageEventArgs): void
- LoginViewModel_LoginSuccess(object, EventArgs): void
- LoginViewModel_LoginFailed(object, EventArgs): void
- MainViewModel_BookEditingStarted(object, BookData bookData)
- BookViewModel_BookEditingFinished(object, EventArgs): void
- MainViewModel_AuthorEditingStarted(object, AuthorEditingEventArgs): void
- AuthorViewModel_BookEditingFinished(object, EventArgs): void
Window
View:BookWindow
-_bookView
-_bookViewModel
-_loginViewModel
-_mainViewModel
ViewModel:BookViewModel
- _model: ILibraryModel
- _editedBookData: BookData
+ BookViewModel(ILibraryModel, BookDat
- OnBookEditingFinished(): void
- Save(): void
- Cancel(): void
«property»
+ EditedBookData(): BookData
+ SaveCommand(): DelegateCommand
+ CancelCommand(): DelegateCommand
«event»
+ BookEditingFinished(): EventHandler
ViewModel:LoginViewModel
- _model: ILibraryMode
- _isReady: Boolean
+ LoginViewModel(ILibraryModel)
- OnLoginSuccess(): void
- OnLoginFailed(): void
- OnExitApplication(): void
- LoginAsync(PasswordBox): void async
«property»
+ IsReady(): Boolean
+ LoginCommand(): DelegateCommand
+ ExitCommand(): DelegateCommand
+ UserName89: String
«event»
+ ExitApplication(): EventHandler
+ LoginSuccess(): EventHandler
+ LoginFailed(): EventHandler
ViewModel::MainViewModel
- isLoaded: Boolean
- _model: ILibraryModel
- _books: ObservableCollection<BookData>
- _selectedBookData: BookData
- _authors: ObservableCollection<AuthorData>
- _selectedBookData: AuthorData
- _covers: ObservableCollection<CoverData>
- _selectedBookData: CoverData
- _revelantLendings: ObservableCollection<LendingData>
- _selectedRelevantLendingData: BookData
- _selectedVolumeData: VolumeData
+ MainViewModel(ILibraryModel)
- OnExitApplication(): void
- OnAuthorEditingStarted(AuthorData, Int32): void
- OnBookEditingStarted(BookData): void
- Model_BookDataChanged(Object, BookData): void
- Model_BookDataDeleted(Object, BookData): void
- Model_AuthorDataChanged(Object, AuthorData): void
- Model_BookAuthorDataCreated(Object, BookAuthorData): void
- Model_BookAuthorDataDeleted(Object, BookAuthorData): void
- Model_CoverDataChanged(Object, BookData): void
- Model_VolumeDataChanged(Object, VolumeData): void
- Model_VolumeDataDeleted(Object, VolumeData): void
- Model_LendingDataChanged(Object, LendingData): void
- Model_LendingDataDeleted(Object, LendingData): void
+ Books(): ObservableCollection<BookData>
+ SelectedBookData(): BookData
+ Authors(): ObservableCollection<AuthorData>
+ SelectedBookData(): AuthorData
+ Covers(): ObservableCollection<CoverData>
+ SelectedBookData(): CoverData
+ RevelantLendings(): ObservableCollection<LendingData>
+ SelectedRelevantLendingData(): BookData
+ SelectedVolumeData(): VolumeData
- LoadAsync(): void async
- SaveAsync(): void async
«property»
+ IsLoaded(): Boolean
+ CreateBookCommand(): DelegateCommand
+ UpdateBookCommand(): DelegateCommand
+ DeleteBookCommand(): DelegateCommand
+ CreateAuthorCommand(): DelegateCommand
+ UpdateAuthorCommand(): DelegateCommand
+ AddAuthorCommand(): DelegateCommand
+ RemoveAuthorCommand(): DelegateCommand
+ CreateCoverCommand(): DelegateCommand
+ AddCoverCommand(): DelegateCommand
+ RemoveCoverCommand(): DelegateCommand
+ CreateVolumeCommand(): DelegateCommand
+ SortOutVolumeCommand(): DelegateCommand
+ DeleteVolumeCommand(): DelegateCommand
+ TurnLendingCommand(): DelegateCommand
+ ExitCommand(): DelegateCommand
+ LoadCommand(): DelegateCommand
+ SaveCommand(): DelegateCommand
«event»
+ BookEditingStarted(): EventHandler<BookData>
+ AuthorEditingStarted(): EventHandler<AuthorEditingEventArgs>
+ ExitApplication(): EventHandler
-_model
-_model
-_model
«interface»
Model::ILibraryModel
-_model
ImageHandler
+ OpenAndResize(String, Int32): Byte[]

A fő ablakhoz tartozó nézet modell kezeli az összes adatmanipulációt, kivéve a könyvek létrehozását, szerkesztését, a szerzők létrehozását és szerkesztését. A kivételeke esetén eseménnyel jelzi a kontrollernek, hogy tegye lehetővé az adatok bevitelét a megfelelő ablak megnyitásával. Van még alkalma a felhasználónak az adatok betöltésére, valamint azok elmentésére. A képek manipulációjához, használ egy külön osztályt [ImageHandler.cs] amellyel elfogadható méretűvé tesszük a képeket, valamint egy konvertert [BookImageConverter.cs]. A releváns kölcsönzési adatok megjelenítéséhez egy másik konvertert használunk [ActiveTypeConverter.cs], amellyel olvasható formátumban tudjuk megjeleníteni, hogy mi a helyzet a kölcsönzéssel.

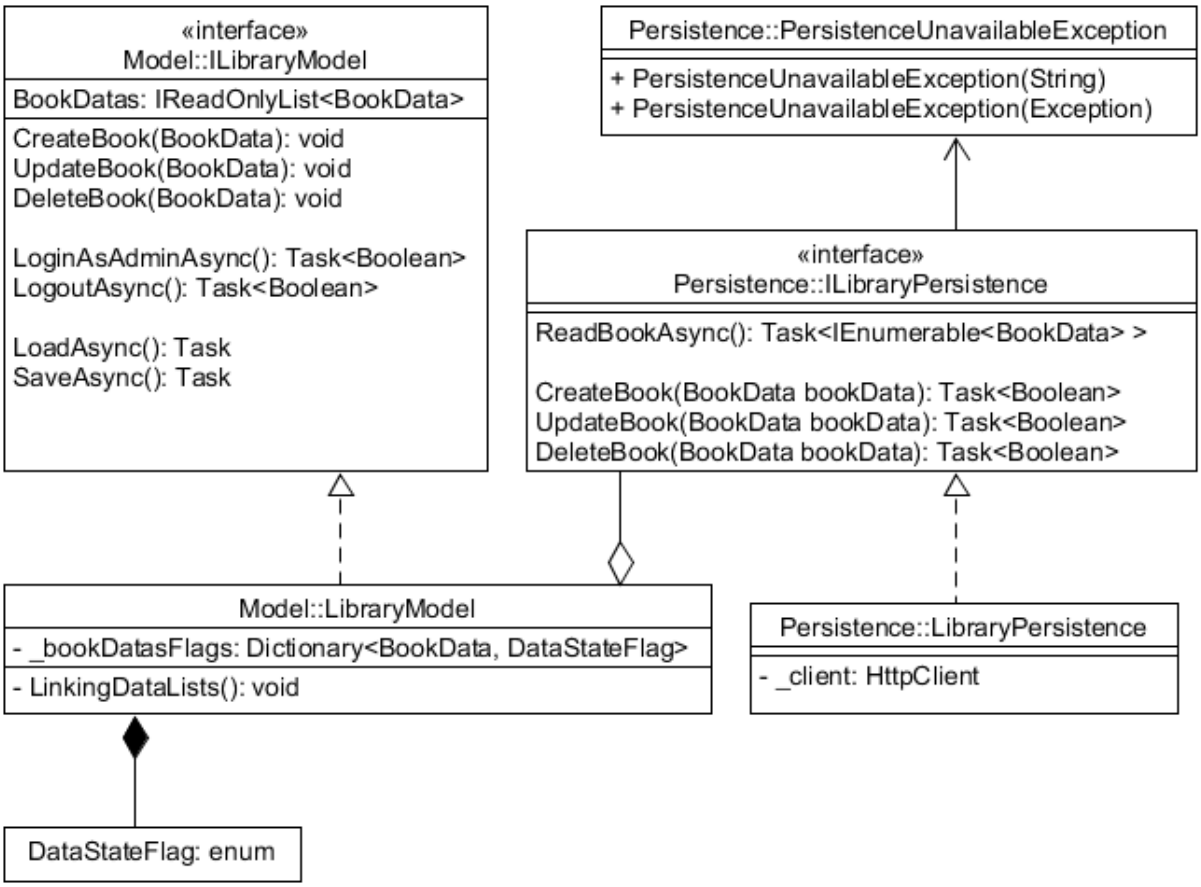
A szerkesztésekhez tartotózó ablakok kezelik az adat bevitelt és ellenőrzést, majd sikertől függően kivált egy lezáró eseményt vagy egy hibaüzenetet megjelenítő eseményt.

Az eseményekhez tartotó eseményparamétereket is itt definiáljuk.

Modell:

A modell [LibraryModel.cs] egy lokális leképezése az adatbázisban levő adatoknak. Az adatok letöltése után, felépít egy lokális adatszerkezetet, melyet, majd a nézet modellek fognak használni és a nézetek megjeleníteni. Minden helyi változtatást külön elmentünk, hogy majd a felhasználó által választott időben visszaszinkronizáljuk az adatbázisba. Data Transfer Objects formátumokat használunk a helyi reprezentációhoz.

Itt is végzünk ellenőrzéseket, valamint még vizsgáljuk, hogy a lokális adatbázis reprezentáción érvényes-e.



Perzisztencia:

A perzisztencia [LibraryPersistence.cs], valósítja meg az WEB API-val történő kommunikációt, mellyel szinkronban tudjuk tartani a helyi adatbázis reprezentációnkat a valódival.

Data Transfer Objects formátumokat használunk a kommunikációhoz.

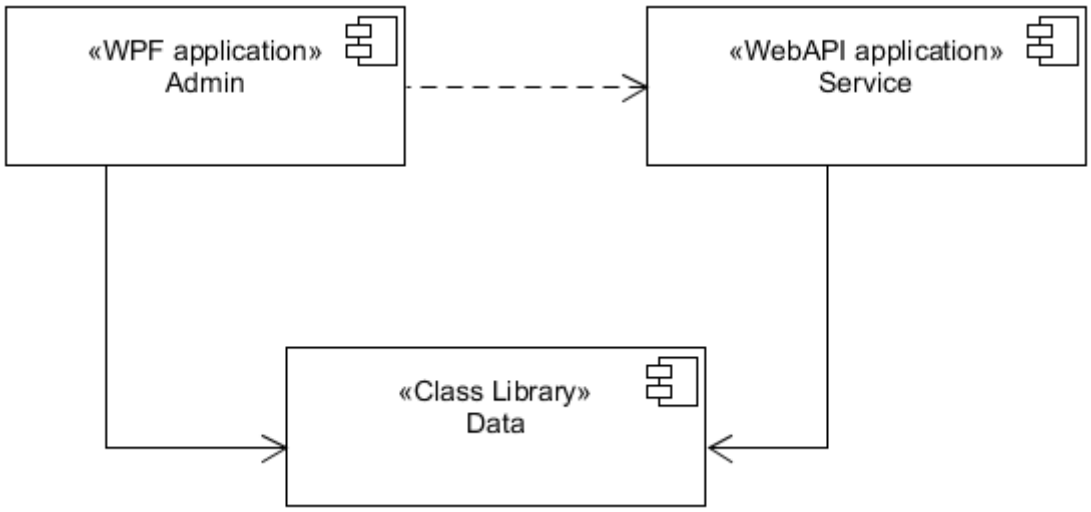
WEB API:

Az API 6 kontrollert tartalmaz:

- [AccountController.cs]
  - `POST: api/Account/LoginAsAdmin`
  - `GET: api/Account/Logout`
- [AuthorsController.cs]
  - `GET: api/Authors`
  - `GET: api/Authors/{id}`
  - `POST: api/Authors`
  - `PUT: api/Authors/{id}`
  - `GET: api/Authors/BookId/{id}`
- [BooksController.cs]
  - `GET: api/Books`
  - `POST: api/Books`
  - `PUT: api/Books`
  - `DELETE: api/Books/{id}`
- [BookAuthorsController.cs]
  - `GET: api/BookAuthors`
  - `GET: api/BookAuthors/{id}`
  - `POST: api/BookAuthors`
  - `DELETE: api/BookAuthors/{id}`
- [CoversController.cs]
  - `GET: api/Covers`
  - `POST: api/Covers`
  - `DELETE: api/Covers/{id}`
- [LendingsController.cs]
  - `GET: api/Lendings`
  - `PUT: api/Lendings/`
  - `DELETE: api/Lendings/{id}`
- [VolumesController.cs]
  - `GET: api/Volumes`
  - `POST: api/Volumes`
  - `PUT: api/Volumes/{id}`
  - `DELETE: api/Volumes/{id}`
  - `GET: api/Volumes/BookId/{id}`

Ezeken a pontokon kommunikál az API. A bemenő paraméterek Data Transfer Objects és megfelelő típusú ID-k. A piros hátterű az amelyet csak adminisztrátor érhet el, a sárgát csak bejelentkezett felhasználó, míg a többit bárki elérheti.

A visszaküldött Data Transfer Objects-ek mindig csak minimális adatokat tartalmaznak.



Tesztelés

A legfontosabb API végpontokat teszteltem csak:

A könyvekért felelős kontrollernek az alap funkcióit néztem meg, jó és hibás adatokkal: összes könyv lekérdezése, könyv létrehozása, könyv frissítése és egy könyv törlése.

A kölcsönökért felelős kontrollernek az alap funkcióit néztem meg, jó és hibás adatokkal: összes kölcsönzés lekérdezése, lekérdezés frissítése és törlése.

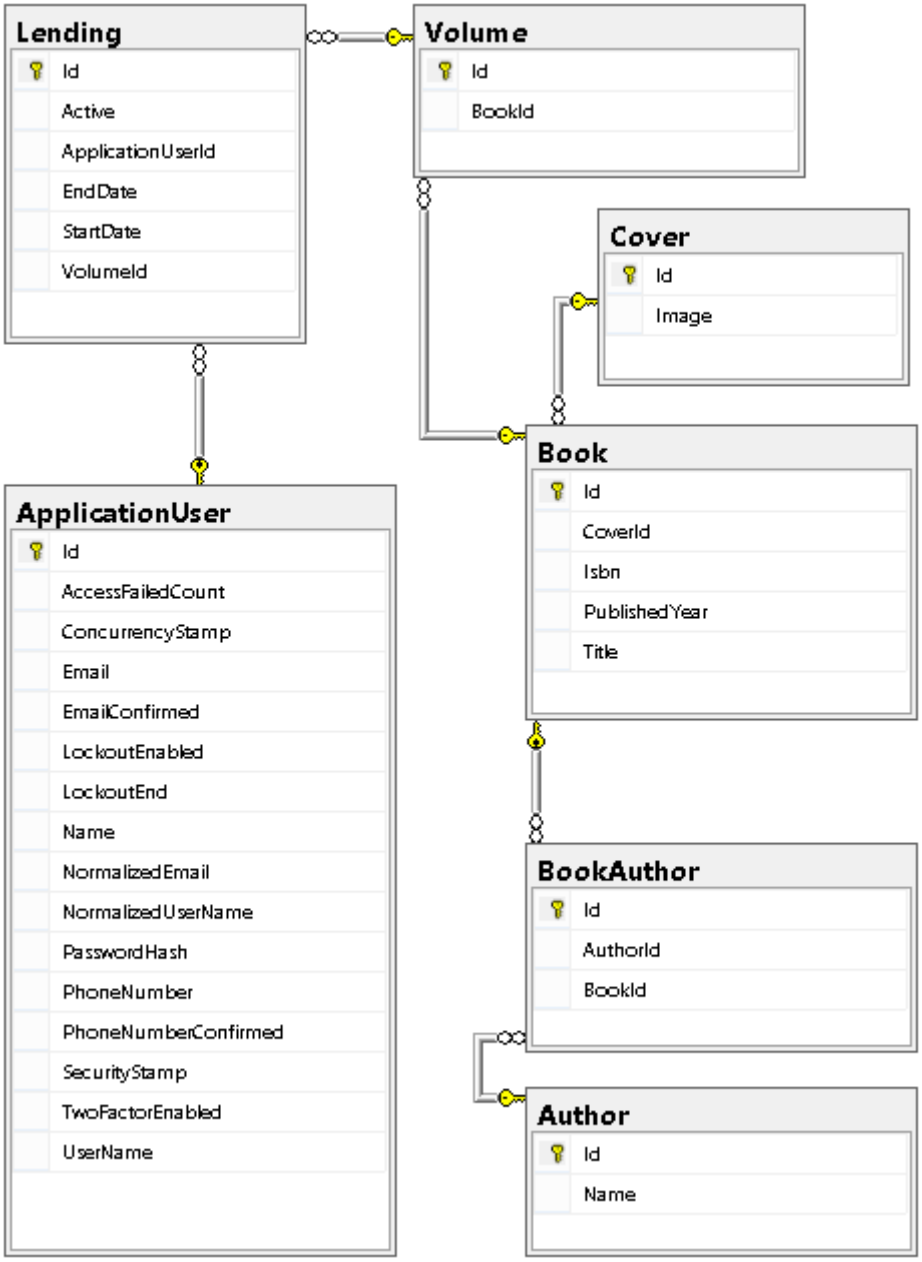
A teszteléshez teszt adatokat használok és memóriában tárolt adatbázist. A kéréseket egy köztes réteggel manipulálom, hogy az API számára úgy tűnjön, hogy egy bejelentkezet, adminisztrátortól jönnek a kérések.

Adatbázis:

A Volume tábla még bővült egy oszloppal: IsSortedOut.

Az ApplicationUser táblához még kapcsolódnak más táblák is melyeket az Identity keretrendszer generált a felhasználó kezeléshez. Ezekkel a ebben a dokumentációban nem foglalkozunk.

Az adatbázist első indításnál vagy adatbázis hiányában létrehozzuk és feltöltjük kezdeti adatokkal, és felhasználókkal (egy adminisztrátorral és két teszt felhasználóval, akiknek lesz kölcsönzéseik).


Lending
Id
Active
ApplicationUserId
EndDate
StartDate
VolumeId
Volume
Id
BookId
Cover
Id
Image
Book
Id
CoverId
Isbn
PublishedYear
Title
ApplicationUser
Id
AccessFailedCount
ConcurrencyStamp
Email
EmailConfirmed
LockoutEnabled
LockoutEnd
Name
NormalizedEmail
NormalizedUserName
PasswordHash
PhoneNumber
PhoneNumberConfirmed
SecurityStamp
TwoFactorEnabled
UserName
BookAuthor
Id
AuthorId
BookId
Author
Id
Name