

- **Telephely:** Ez jelöli, hogy az adat az melyik telephelyhez tartozik az adatbázisban. Ez vonatkozik az iktatásra és a törzsadatokra egyaránt.
- **Felhtelephely:** Minden felhasználóhoz tartozik egy vagy több telephely ahová tud iktatni vagy törzs adatokat rögzíteni.
- **Partner:** Azokat a partnereket tartalmazza akiket a
- **Partnerügyintéző:** A megadott partnerhez tartozó ügyintézőket tartalmazza
- **Partnerügyintéző kapcsoló:** Az a személy, intézmény vagy cég aki küldte az iratot. Munkaszerződéseknél a partner a dolgozó nevét jelöli. Lehet például E-on, Járási hivatal..stb. Ennek a táblának az id-je fog idegen kulcsként megjelenni az iktatásban.
- **Csoport:** Az iratok azon típusait jelöli, amely egységhez kapcsolódik az iktatandó anyag. Például Ellátotti, Főzőkonyha, Munkaügy.
- **Jelleg:** A dokumentum formai megjelenésének megadása. Ez lehet e-mail, küldemény, fax, levél, munkaügyi irat.
- **Ügyintéző:** Ez a szervezetten belüli dolgozó kollégára utal, hogy ezt az ügyet vagy iratot ki intézi.
- **Doc:** Itt tároljuk az iktatásokhoz feltöltött állományokat mediumblobban. Illetve eltávolítjuk még annak nevét és a kiterjesztését is. Lehet Pdf,JPG,PNG,XLSX,DOCX...stb.
- **Ikonyv docs:** Az adott iktatáshoz tartozó dokumentum.
- **Ikonyv:** Ez maga az iktató könyv. Ha bejön egy irat

vagy kimegy azt itt lesz rögzítve. Az iktatószámot tárolt eljárással fogom előállítani ami a megadott adatok alapján fog generálódni. Egy példa: B-SZ/R/3/2019 Ennek felépítése

- Az első karakteret az határozza meg, hogy K - kimenő vagy B - bejövő
- A második karakter a jellege határozza meg SZ pl szerződés.
- A harmadik karakter a telephely jelöli pl. R - Rákóczi, V- Vajda stb..
- A negyedik karakter sorozat a sorszám ami lehet kötőjeles Pl. B-SZ/R/3-1/2019 vagy B-SZ/R/3-1-1/2019 a válaszokhoz mérve.
- Az utolsó rész pedig az évet jelöli.

A generált iktatószám el is lesz tárolva a táblában.

2) **Nézetek:** Egyenlőre három nézetem van. Első a creatIktSzam ahol a iktató könyv tábla összes idegen kulcsa össze van kapcsolva a táblájával és ezekből összeállítja az aktuális iktató könyv iktatószámát. A második a currentYearIkonv itt lekérem az aktuális év iktatásait olyan formában amiben majd a programban kelleni fog. Illetve az utolsó a prevYearIkonv ami hasonló a currentYearIkonvhoz de itt saját kezűleg kell paraméterezni az évet.

3) **Tárolt eljárások:** Minden kérést amit lehetett tárolt eljárásba tettem. Ezzel is megakadályozva az SQL injectionnek a lehetőségét. Összesen 39 darab lett belőlük, de ezeknek a száma valószínűleg csak növekedni fog az iktató program fejlesztése során mivel előfordulhatnak olyan lekérdezések amikre még nem gondoltam. Pár fontosabb eljárást fogok csak bemutatni, mivel a nagy része csak az adatok megfelelő módosításáról, törléséről és hozzáadásáról szól. A legfontosabbak a következők. AddRootIkonv, AddSubIkonv, DelIkonv, setDeletedByValaszID, getNextIktatottID, getIkonvek. Ezek okozták a legtöbb fejtörést a számomra.

**AddRootIkonv:** Bemenete az iktatóhoz szükséges adatok mint a Tárgy, Hivatkozási szám, Ügyintéző id-je, PartnerÜgyintézőkapcsoló id-je, stb... Első lépésben lekérem az aktuális évet, erre azért van szükség hogy az iktatószámhoz hozzá tudjam adni. (Az aktuális év nem feltétlenül egyezik meg az aktuális dátummal). Második lépésben lekérem a következő Id-t. Mivel több telephely is szerepel a táblázatban ezért van erre szükség, így mindegyik telephelyen a számozás konzisztens marad illetve még a válasz iktatások is bezavarnának, hogy ha csak a sima primary key-t használnám. Harmadik lépésben hozzáadom az új iktató könyvet még iktatószám nélkül. Negyedik lépésben lekérem az ő ID-ját a táblából így az generateIktSzam viewval le tudom generáltatni az iktatószámát és hozzáadni az iktatókönyvhöz. Utolsó lépésben vissza adom az új iktatásnak az id-ját így ha kell le tudom egyből kérdezni. Ez lehet változni fog, mivel valószínűleg csak az iktatószámra lesz szükségem a hozzáadás során.

**AddSubIkonv:** A bemenete hasonló az szülőhöz de itt még paraméterként várom annak az iktatókönyvnek az id-jét

amihez ez az iktatás kapcsolódik. A lépések ugyan azok mint a szülő ikönyvnél, de itt mikor lekérem az iktatókönyvnek a következő id-jét még hozzá kell tennem a szülőnek az ID-jét is. Iktatószám generálása kicsit másképp történik. Először lekérem az szülő iktatószámát és azt átalakítva mentem el az ikönyv táblában.

**DelIkonv:** Az iktató könyv törlése elég macerás dolog. Mivel a könyvnek lehetnek gyermek iktatásai és a gyermek iktatásnak is lehetnek gyermek iktatásai, ezért erre oda kell figyelnem, hogy ha az egyik szülőt kiszedjük a fa struktúrából akkor az összes gyermeket is "törölje". Természetesen végleges törlést nem csinálunk csak a deleted flaget 1-re állítjuk. Bemenő paraméterek: az iktató könyvnek az id-je és a felhasználónak az id-je. Igazából nem csinál mást csak meghívja a setDeletedByValaszID-t a bemenő paraméterekkel.

**setDeletedByValaszID:** A bemenő paraméterek ugyan azok mint a DelIkonvnek. Első lépésben megnézzük, hogy az iktatásnak vannak-e gyermekei, ha nincs akkor a flaget 1-re állítjuk ha van akkor a következő lépéseket kezdjük. Amíg van gyermeke addig kiválasztjuk az első gyermeket. A gyermek delete flagjét 1-re állítjuk és meghívjuk ezt az eljárást a jelenlegi gyermek iktatásra. A legvégén az eredetileg meghívott iktatókönyvnek is a flag-jét 1-re állítjuk.

**getNextIktatottID:** A bemenete a telephelynek az id-je és hogy ha van akkor a szülőnek az id-je. Először megnézem hogy a válasz id null-e ha igen akkor a következő selectet meghívom: select IFNULL(MAX(iktatottid),0)+1 into nextiktatottid from ikönyv where telephely = telephely\_b and valaszid is null Viszont ha van valaszid akkor ugyan ezt a selectet hívom meg csak a where feltétel módosul.

**getIkonvek:** Bemenete a userid így kitudom keresni hogy a felhasználóhoz melyek azok a telephelyek amik hozzá tartoznak és csak azokat fogom lekérni. Egy ilyet még írnom kell a prevYearIkonv-re is de ahhoz még kelleni fog bemenő paraméternek az év. Illetve a lapozáshoz még fog kelleni egy tól-ig paraméter is.

## B. gRPC és Protocol Buffers

### Grpc:

A gRPC rekurzív mozaik szó. A jelentése gRPC Remote Procedure Call. A google fejlesztette 2015-ben. Nyílt forráskódú. Két főrészből áll. Első a gRPC protokoll és a ProtocolBuffer azaz a adat szerializáció. A protokoll http2 alapokon működik és ki is használja azok előnyeit. Például fejléc tömörítés, egyetlen folyamatos TCP kapcsolat, megszakító és időtúllépés contract a kliens és a szerver között. [1] A gRPC RPC típusa: Unáris: Azaz a kliens küld egy requestet és a szerver válaszol arra. Kliens streaming RPC: Ahol a kliens több üzenetet küld ezt megvárja a szerver és ha a kliens végzett a szerver válaszol egy válaszban. Server Streaming RPC: Ami a Kliensnek az ellentettje. És van a bidirectional streaming rpc ahol a kliens és a szerver egymástól függetlenül tudnak több üzenetet küld egymásnak. [3]

**Protocol Buffer:** A protocol buffer egy nyelv és platform semleges protokoll, ami az adatok szerializációjáért

felel. Sokkal egyszerűbb, hatékonyabb és flexibilisebb mint egy XML struktúra és könnyebben olvasható is. [2] Két részből áll az egyik a Message a másik a Service amiben az rpc-k találhatók. Megtalálhatóak benne skalár típusok Pl.: int32,int64,double,string stb ...

1) *Message*: A message egy üzenet ami az rpc híváskor adható meg paraméterként illetve ilyen üzeneteket tud viszásküldeni a szerver. Itt lehet a classokat kialakítani illetve egyéb segéd osztályokat. Illetve az rpc hívásoknál nem lehet üres a metódus ezért létre kell hozni egy EmptyMessage aminek nincs semmi mezője és ezt beírakni. Nekem egyelőre csak a user-hez kapcsolódó rpc-k és messagek vannak kialakítva. Ami LoginMessage, Answer, User, Privilege. Az LoginMessage tartalmaz egy string felhasználó nevet és egy string password mezőt. Ezt fogja a kliens elküldeni a szerver felé majd az válaszol egy User modellel. Az Answer tartalmaz egy bool error és egy string message mezőt. Ami a kijelentkezéskor és a regisztrációkor használatos.

2) *Service*: A service nem más mint azoknak a rpc metódusoknak gyűjtőneve amit használni szeretnénk a grpc-kor. Több service is megadható, de ki kell választanunk szerver indításakor, hogy melyiket használja. Egyelőre négy metódusom van a Login, Logout, Register, ModifyUser. Ezek lesznek a userrel való műveletek. Mint látszódik is a login egy LoginMessage-t a többi User Message vár.

A .proto fájl:

```
syntax = "proto3";

package Iktato;

service IktatoService {
    rpc Login (LoginMessage) returns (User);
    rpc Logout (User) returns (Answer);
    rpc Register (User) returns (Answer);
    rpc ModifyUser (User) returns (Answer);
}

message LoginMessage{
    string username = 1;
    string password = 2;
}

message EmptyMessage{

}

message Privilege{
    int32 id = 1;
    string name = 2;
}

message Answer{
    bool error = 1;
    string message = 2;
}
```

```
message User {
    int32 id = 1;
    string username = 2;
    string fullname = 3;
    string password = 4;
    Privilege privilege = 5;
}
```

### C. Szerver és Kliens

1) *Szerver*: A szervert is c#-ban írom bár lehetne nagyjából bármilyen nyelven, mivel a választott technológiának. Egy WPF alkalmazást kezdtem el írni, de valószínűleg ezt módosítani fogom egy console alkalmazásra. Egyelőre egy pár metódus van még csak implementálva, hogy a kliens felé tudjunk küldeni valamilyen adatot. A későbbiekben még implementálom a logolást, JWT token hitelesítést. Biztonság tekintetében implementáltam az ssl kapcsolatot. Generáltam egy public és private key párt openssl segítségével. Utána a public keyt hozzáadtam a klienshez, míg a private és public -t a szerverhez. A fájlokat beolvastam és a channel constructorába hozzá kell adni. [4]

2) *Kliens*: A kliensen már telepítettem a caliburn.micro és a grpc részeit, hogy működjenek a hívások. Az caliburn.micro egy kicsi, ám de erős keretrendszer, arra lett tervezve, hogy applikációkat készítsenek vele az összes XAML platformon. Erős támogatása van a MV\* típusú tervezési mintákhoz ezáltal segíti a gyors fejlesztés anélkül, hogy feláldoznánk a kód minőségét vagy tesztelhetőségét. [5] A loginnak a fejlesztésével kezdtem, ennek a nagyja már megvan még a hiba kezelés nem megoldott benne, illetve a server által későbbiekben jwt token tárolása illetve a gRPC hívások sem mivel minden hívás előtt be kell állítani majd a header részében a tokenet. Ezért erre egy külön proxy classt fogok csinálni, hogy a későbbiekben azzal ne kelljen foglalkoznom.



Fig. 2. A kliens login felülete.

## FELADATOK, TERVEK ÉS AZOK KIVITELEZÉSE A SZAKDOLGOZAT BEFEJEZÉSÉIG.

Lényegében az egész egész szoftver fejlesztése még hátra van, de szerencsére a caliburn.micro és a gRPC segítségével nem lesz nehéz időben ezt befejezni. Tervbe van még véve, hogy TDD-ben lesz a fejlesztés és egy loggolás server és kliens oldalon is. Illetve, hogy a RPC hívásoknál várni fogok még valami token -t is ami validálja a usert hogy meghívhatja e az adott metódust.

## EREDMÉNYEK

Ebben a félévben elértem, hogy az adatbázissal ne nagyon kelljen foglalkoznom már a fejlesztés során illetve sikerült összehozni egy körülbelüli képet a munkáltatómmal, hogy hogyan is nézzen ki a program. Sikerült SSL titkosítás implementálása, így a gRPC szerver és a kliens nem egy insecure channelen kommunikálnak.

## REFERENCES

- [1] <https://grpc.io/docs/>
- [2] <https://grpc.io/docs/talks/>
- [3] <https://blog.container-solutions.com/introduction-to-grpc>
- [4] <https://bbengfort.github.io/programmer/2017/03/03/secure-grpc.html>
- [5] <https://caliburnmicro.com/>