Matematikai és Informatikai Intézet

Iktató program

Készítette
Pekár Mihály
Pti vagy Szofi?

Témavezető
Balla Tamás
Tanársegéd

Eger, 2020

Tartalomjegyzék

1. fejezet

Bevezetés

Szakdolgozatom témájaként a munkám során mindennap előforduló iratkezelési problémát szeretném megoldani. Az intézmények a nyomon követhető háttérmunka nélkül nem képesek működni, ezért a napi munka során keletkezett dokumentáció szakszerű kezelése és iktatása nélkülözhetetlen. Mivel minden közfeladatot ellátó szerv, intézmény, vállalat más és más tevékenységet végez, ezért számukra elengedhetetlen a pontos, precíz iratkezelés, amit szabályozott és rendszeres ügyviteli munkával érhetnek el.

2018 évében egyes állami fenntartásban lévő szociális szférában működő szervezeteket egyházi tulajdonba kerültek. Ennek következményében a Szociális- és Gyermekvédelmi Főigazgatóság fennhatósága alatt álló Békés Megyei Körös-menti Szociális Centrum szarvasi telephelye a Magyarországi Evangélikus Egyház szarvasi diakóniájának részévé vált. lett. Mivel egy időben több telephellyel is bővült, ezért szükségessé vált az ügyviteli, leltározási és nyilvántartó rendszereinek felülvizsgálata. Az eddig 50 fővel működő diakónia létszámához elegendő volt az Excel használata, viszont a megtöbbszöröződött létszámmal a dokumentumok átláthatatlanabbak, kezelhetetlenebbek lettek és egyre több problémát okoztak az elszámolásokban, kimutatásokban, zárásokban. Ekkor keresett meg a gazdasági vezető olyan céllal, hogy e munkafolyamatokat rendszerezzük, szabályozzuk és egyszerűsítsük le szoftverek segítségével. Programozói feladataim között szerepelt munkaruha, leltár és telephelyek ételadag megrendeléseit nyilvántartó rendszerek fejlesztése. Számomra ez egy jó lehetőséget adott, hogy a tanulmányaim során szerzett programozói tudást gyakorolhassam. Megragadtam az alkalmat és minden egyes programhoz más és más programozási nyelvet alkalmaztam. Sok segítséget nyújtott számomra, hogy ezek nagy részét már lehetőségem volt kurzus keretei között megtanulni az egyetemen. Például a munkaruha nyilvántartóhoz HTML, JavaScript és CSS-t, keretrendszernek node.js és bootstrap-t választottam, ezek nagyban megkönnyítették a fejlesztést. A leltár programhoz a java Spring-keretrendszerét alkalmaztam, de kipróbáltam az Excel-ben a VBA programozási nyelvet is.

Ezen programok fejlesztése után jutottunk el az iktatáshoz, amelyet ugyanúgy Excel-ben végeztek a kollégák. A vezető mindenképpen le szerette volna cserélni ezeket

a munkafüzeteket, mert teljesen alkalmatlanokká váltak a megnövekedett mennyiségű iratok kezeléséhez. Arra jutottunk, hogy ezeket a munkafolyamatokat egy erre fejlesztett programban kellene végezni. A várható pozitív hatások, úgymint az egyszerűbb áttekinthetőség, a gyorsaság és a költséghatékonyság megerősített minket a szoftver létrehozásában. Arról még említést se tettünk, hogy egyre több irat érkezik valamilyen elektronikus formában és ezeknek a tárolása, megőrzése nem egyszerű. Első és legfontosabb vágy, kritérium az volt, hogy ebben a programban nyomon követhessék az összes telephelyen keletkező iktatást, továbbá az is fontos követelmény, hogy a dokumentumok könnyen és pontosan visszakereshetőek legyenek.

Így a dolgozatom célja munkáltatóm ügyviteli rendszerének összehangolása, átszervezése, iktatásának központosítása és digitalizálása. Azért hangsúlyos ez a terület ebben az intézményben, mert az itt folyó szociális és ápolói munka során számos egyéb hivatalos szervvel kerülünk kapcsolatba, ami megköveteli a naprakész, pontos és folyamatában követhető iratkezelést.

2. fejezet

Követelmény specifikáció

2.1. Fogalomtár

iktatókönyv: Olyan nem selejtezhető, hitelesített iratkezelési segédeszköz, amelyben az iratok iktatása történik: a be- és kimenő iratok nyilvántartásba vétele, iktatószámmal történő ellátása, dátum és partner megjelölésével.

user: Az a személy, aki az adott funkció használatára jogosult és a funkció működtetése során számára ismertté váló vagy módosítható adatokhoz rendelkezik a betekintési vagy módosítási jogosultsággal.

admin: Az a személy, akik a user jogosultságnál magasabb hozzáférési szinttel rendelkeznek.

rendszergazda: Az a személy, aki az iktatóprogram telepítésével, karbantartásával, rendszernapló gondozásával, valamint a szerver működésének ellenőrzésével foglalkozik.

szerver: Olyan nagyteljesítményű számítógép vagy szoftver, ami a más számítógépek számára a rajta tárolt vagy előállított adatok felhasználását a szerver, hardver erőforrásainak kihasználását, illetve más szolgáltatások elérését teszi lehetővé.

adatbázis: Azonos minőségű többnyire strukturált adatok összessége, amelyet egy tárolására, lekérdezésére és szerkesztésére alkalmas szoftvereszköz kezel.

iktatás: Iratnyilvántartás alapvető része, amelynek során a beérkező iratot, illetve a keletkezett iratot iktatószámmal látják el.

partner: A szervezet azon résztvevői, akik felé hivatalos iratok készülnek. Ez lehet cég, magánszemély vagy más hatóság.

partnerügyintéző: Az a személy, aki a partnernél hivatalos irattal foglalkozik.

telephely: Az intézmény különböző földrajzi helyeken elhelyezkedő egységei.

jelleg: Az iktatott anyag formája. Lehet fax, e-mail, levél, munkaügyi irat stb.

iktatásicsoport: Ügyirat besorolását, osztályozását lehetővé tevő zárt adatkészlet, amelyhez az adott szervnél ügyirat hozzárendelhető.

ügyintéző: Az a személy, aki az ügyirat nyilvántartásba vételével foglalkozik.

irány: Azon jellemző, amely megmutatja, hogy az intézmény felé érkezett vagy általa küldött az irat.

törzsadat: Azon adatok összessége, melyekből az iktatás összeáll.

logolás: Olyan adatok rendezett összessége, amely az események rekonstruálására alkalmas, részletességben tárol információt az iratkezelési eseményekről és egyéb tevékenységekről.

irat: A köziratokról, a közlevéltárakról és magánlevéltári anyag védelméről szóló 1995. évi LXVI. törvény 3.§ c pontja szerinti adat együttes.

előzményezés: Az iratkezelési folyamat azon művelete, amely során megállapításra kerül, hogy az új irat egy már meglévő ügyirattal kapcsolatban áll-e, vagy az új iratot új ügy első irataként kell-e nyilvántartásba venni.

hivatkozási szám: A beérkezett irat iktatási száma.

jogosultság: Egy adott művelet elvégzésének lehetősége az iktatóprogramban, egy adott személyre vagy rendszerelemre vonatkozóan, szerepkör hozzárendelése.

2.2. Jelenlegi helyzet leírása

Minden hivatalos iratokkal foglalkozó kollégánál van egy Excel tábla, amit iktatásra használ. Ebben a táblában iktatja a hozzá beérkező, illetve az általa elkészített ügyiratokat. Hátrányai ennek a helyzetnek, hogy az iktatások állandósága sérül, azaz egy-egy irat kaphat olyan számot, ami már foglalt. Mivel vannak, olyan esetek, amelyek több munkacsoportot is érintenek, az adott irattal előfordulhat, hogy többször vagy egyszer sem iktatnak le. Az iktatásokhoz nem lehet előzményezést rendelni. Az egységek nem látják egységesen az iktatási anyagaikat, így egymás segítségére szorulnak. Az iktatott dokumentumokat csak több perces keresés árán tudják megtalálni. Excel hibái a jelenlegi helyzetre:

1. A legtöbb embernek nincs kellő ismerete:

   Valamilyen oknál fogva az emberek többsége nem szereti az Excelt, főleg azon dolgozók között, akiknek nincs tapasztalata a program használatához. Legtöbben zavarónak és

félelmetesnek találják, ezért meg sem próbálják megtanulni a használatát. Ez nálunk sincs másképp, hasonló problémákba ütközök nap, mint nap a kollégáim körében. Ha valaki kap egy feladatot, amit Excelben kell elvégeznie a legapróbb nehézségnél is segítséget kérnek.

2. A fontos adat eltűnik

   Mivel az összes adatot látjuk egyszerre, ezért nagyon nehéz átnézni úgy, hogy felfedezzük melyik adat fontos és melyik nem. Sokszor tapasztalom azt a kollégák körében, hogy a használt Excel dokumentumokat több percig is nézik, hogy megtalálják a keresett adatot, amire kíváncsiak voltak. Természetesen lehet használni színezéseket is, de e módszerek használata egy idő után növelik az átláthatatlanságát. Továbbá a nem egységesen használt színkezelés csak bonyolítják az értelmezését.

3. Nyomonkövetés

   Sok esetben előfordult, hogy a dolgozók egymás adatait véletlenül törölték, módosították. Mivel a munkafüzetet nem arra tervezték, hogy eltárolja a változásokat, ezért ha egy adatot módosítunk vagy törlünk, azt csak mentésből tudjuk visszaállítani. Ezeknek a manipulációknak a visszakövetése nem lehetséges.

4. Megosztás

   Annak ellenére, hogy léteznek felhőalapú technológiák, a mai napig nehéz megosztani a munkafüzetet több dolgozó között. Mivel annak a lehetősége, hogy az adatot kitörlik vagy megváltoztatják, a munkafüzetet ritkán tesszük elérhetővé mások számára valós időben. A legjobb megoldás az lehet, hogy heti rendszerességgel elküldjük e-mail formátumban. Viszont ez meg azt a problémát veti fel, hogy a fontos információ elveszhet a dolgozók beérkezett üzenetei között.

5. Az adatkezelési szabályok betartása

   Az Excelben nem lehet az iktatószámokat nyomon követni és azok konzisztenciáját megőrizni. Könnyen el lehet rontani az iktatószám felépítését. Több telephely iktatásának tárolása a munkafüzeten komplikációt okoz, illetve nincs kontroll az iktatások felvitelében, így bárki bármikor ütközhet. A dokumentumkezelés is nehézséges, az iktatásokhoz külső link formájában hozzáadhatjuk a dokumentumot, viszont ha ezek az adatok más helyre kerülnek, akkor az összes csatolt linket módosítani kellene.

6. Nem célszoftver

   Ezt a programot nyilván nem iktatás, ügyiratkezelés folyamataira tervezték, az adatmennyiség és ezen folyamatok bonyolultságának növekedésének köszönhetően a szoftvernek a használata egyre kellemetlenebbé válik.

7. Jogosultság

   A felhasználók kezelésére és azoknak a jogosultságának szabályozására sincs lehetőség. Így bármilyen adaton bárki tud módosítást végezni. Már pedig ez számunkra nagyon fontos funkció, hogy ezeket a folyamatokat felhasználó szinten tudjuk befolyásolni, hogy az adatok biztonságban legyenek.

2.3. Vágyálom rendszer

Egy olyan központosított iktatás létrehozása, ahol a személyenkénti gondolkodásmódot elhagyhatjuk. A gazdasági csoport vezetője szeretné látni az összes telephely iktatását, mivel a jelenlegi helyzetben ez nem megoldott. Feleljen meg az iratkezelésre vonatkozó jogszabályoknak és az intézményi ügyiratkezelési szabályzatnak. A legfőbb cél, hogy az iratkezelés legyen szakszerű, gyors, gazdaságos és naprakész.

Tudjon felhasználókat kezelni a program, azaz legyen lehetőség a userek hozzáadására, módosítására és inaktívvá tételére. Két jogosultsági szint legyen, sima felhasználó és admin. A jelszavakat az adatbázisban titkosítva tárolja. El lehessen különíteni telephelyeket, illetve legyen olyan lehetőség is, hogy egyes felhasználókhoz több telephely is tartozzon vagy mind.

Az iktatás összeállítása egyszerű és könnyen kezelhető, valamint az iktatószám azonnal olvasható legyen. Ne kelljen ügyelni arra, hogy ki és mikor iktat, a program automatikusan generálja a megadott adatokból az iktatószámot. Ez úgy generálódjon, hogy vegye figyelembe a felhasználó által felvitt adatokat és az aktív évet. Minta az iktatószámra: B-M/R/2/2020, az első betű reprezentálja az iktatásnak az irányát, a kötőjel utáni rész az iktatási csoportot jelölje, ami maximum 3 karakter hosszúságú lehet. Az „R” betű jelöli a telephely kezdőbetűjét, ebből is tudni fogjuk, hogy az iktatás melyik telephelyhez tartozik. Ezután következik a sorszám, ami a soron következő iktatást jelöli a telephelyen. Az utolsó szakasz az aktív évet jelölje. Illetve szeretnénk egy olyan eljárást is, ahol a felhasználó kiválaszthatja az iktatás előzményét. Ezáltal az iktatószámot oly módon befolyásolja, hogy a korábban iktatott számnak soron következő alszámát generálja le (előzményezés). Például: B-M/R/2-1-2/2020. Évenként elkülöníthetők és lezárhatóak legyenek az iktatások. Irányuk lehet bejövő és kimenő jellegű.

Törzsadatokat egy menüpontban legyen lehetőség kezelni. A felhasználók számára legyen elérhető partner, ügyintéző, partner ügyintézője, jelleg adatok kezelése. Adminnak ezeken kívül legyen még lehetősége a csoport, telephely, felhasználók, év zárás/-nyitás módosítására, felvitelére. Viszont a felhasználók jelszavát csak a rendszergazda módosíthatja.

A dokumentumot gyorsan és egyszerűen fel lehessen tölteni az iktatásokhoz. Ezek jellege Word, Excel, PDF. Könnyen lehessen keresni a felvitt adatok között. Egyes

iktatásokhoz több dokumentum is csatolható legyen. Ezeknek a dokumentumoknak a megtekintésére, letöltésére és törlésére legyen lehetőség.

A rendszergazdának legyen szerveren kezelő felülete, ahol az adatbázist, felhasználókat és a naplózást könnyen tudja kezelni. Itt legyen meg az a lehetősége, ahol a felhasználók jelszavát módosíthatja.

Jó lenne az egyházi jelleg megjelenése a programban. Például a főoldal tartalmazhatná a napi igét, illetve az evangélikus egyház jelképe, azaz a Luther-rózsa is megjelenhetne.

2.4. A rendszerre vonatkozó pályázat, törvények, rendeletek, szabványok és ajánlások felsorolása

(ok, hogy ismertetjük a jogszabályi hátteret, de akkor a törvények felsorolása mellett röviden definiáljuk, hogy mit vár el ... ne csak azt mondjuk, hogy figyelembe kell venni, mert ez így elég logikátlan) (latex-ben ki lesz dolgozva) Az iratkezelés menetét alapvetően meghatározza a jelenleg hatályos, többször módosított 1995. évi LXVI. törvény, a levéltári törvény a köziratokról, a közlevéltárakról és a magánlevéltári anyag védelméről. CompLex Jogtár, 1995. évi LXVI. törvény, a levéltári törvény a köziratokról

335/2005. (XII. 29.) Korm. rendelet a közfeladatot ellátó szervek iratkezelésének általános követelményeiről CompLex Jogtár, 335/2005. (XII.29.) Korm. rendelet

Ügyiratkezelési intézményi szabályzat:...

2.5. Követelmény lista

A fent említett igények alapján állítottuk össze a követelmény listát, ami megfogalmazza, hogy egyes verzió számú program részek már milyen tudással bírjanak.

ID	Név	V.	Kifejtés
1	Bejelentkezés és jelszó védelem	0.1	Legyen egy szép és egyszerű bejelentkezési felület, ahol a felhasználó nevet el lehessen menteni, de csak akkor, ha szeretné a user. A felhasználók jelszavát a rendszergazda se tudja visszanézni, viszont módosítani csak ő tudja. A jelszavak legyenek titkosítva az adatbázisban.
2	Főoldal és ige	0.2	A belépés után jelenjen meg az üdvözlő felület. Ahol megtalálható az egyház szimbóluma, és a napi ige.

3	Törzsadatok kezelése	0.3	Egy helyen lehessen kezelni a törzset. Mindegyik törzshöz legyen lehetőség hozzáadni, törölni és módosítani. Minden telephelyhez külön törzs adatbázis tartozzon, ne keveredjen a többi telephellyel. A felhasználó jogosultsággal rendelkező munkatárs ne tudjon iktatási csoportot, telephelyet, felhasználót és évet kezelni.
4	Iktatás kezelhetősége	0.4	Az iktatáshoz tartozó információk egy legördülő menüben kiválaszthatók legyenek(ügyintéző, jelleg, csoport, stb. . .). A tárgynak és hivatkozási számnak legyen külön beviteli mezője, csak aktuális évre lehessen iktatni. Előzményezési lehetőség. Az iktatásokat törölni csak admin jogosultsággal. Az iktatások bárki által módosíthatók, de csak azokat az információkat, amik nem befolyásolják az iktatószámot. Két dátum, érkezett és határidő lehessen megjegyzést fűzni az iktatáshoz. A hivatkozási szám, megjegyzés és a partnerügyintézőn kívül minden mezőt kötelező kitölteni.
5	Dokumentumok kezelése	0.5	Minden iktatáshoz lehessen feltölteni dokumentumokat. Dokumentumok jellege lehet PDF, Word, Excel. Ezeket egyszerűen vissza lehessen nézni, és ha kell törölhetőek legyenek.
6	Keresés az iktatások között	0.6	A keresés szűrhető legyen évre és irányra. De az iránynál az összes is látszódhasson. Keresni lehessen iktatószámra, tárgya, partnerre, jellegre, ügyintézőre, csoportra és hivatkozási számra. A keresésnél is lehessen feltölteni dokumentumot és módosítani az iktatást.
7	Naplózás beállításai	0.7	A szerver oldalon állíthatóak legyenek a logolás szintjei. Egyszerűen lehessen módosítani a log fájl elérési útját.

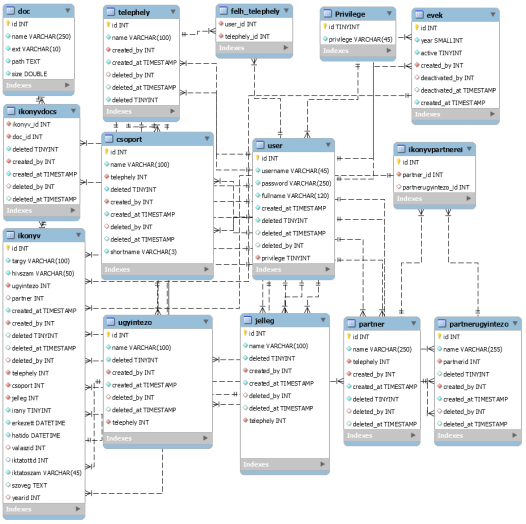
8	Adatbázis mentés	0.8	A szerveren legyen lehetőség az adatbázis mentésére és annak visszatöltésére. Az adatbázis mentés ütemezhető legyen.
9	Felhasználó kezelés	1	A szerver oldalon is legyen lehetőség a felhasználó felvitelére, módosítására és törlésére. A jelszavak módosítása csak innen legyen lehetséges.

2.6. Technikai feltétel

A rendszer MySQL adatbázis kezelőt használjon, a program szerver részét lehessen futtatni Windows Szerver 2012R2. A kliens program Windows platformon legyen használható. Interneten keresztül is lehessen kommunikálni a szerverrel. A kapcsolata kliens és a szerver között egy titkosított adatfolyamon történjen.

2.7. Rendszerspecifikáció

Adatbázis terv: Az adatbázis a következő táblákat fogja tartalmazni: Ikonyv, Ikonyv-

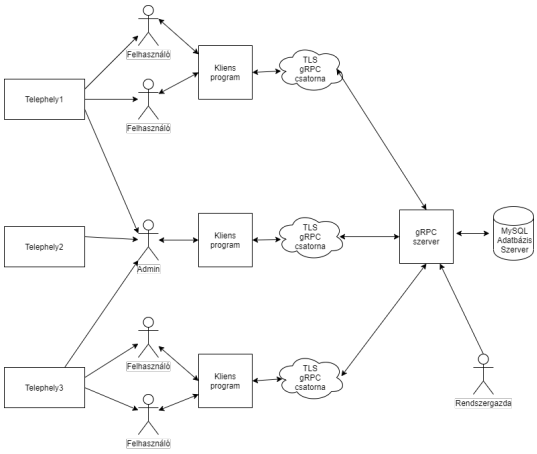


2.1. ábra. Adatbázis tervezet

docs, Doc, Telephely, Csoport, Ugyintezo, Jelleg, User, felh_telephely, Privilege, evek, ikonyvpartnerei, partner, partnerugyintezo. Ezek a táblák lefedik a követelmény specifikációkban foglaltakat. Minden adatbázis művelet tárolt eljáráson keresztül végződik.

A program működésének ábrázolása use case diagram segítségével:

Az egyes telephelyeken lévő felhasználók bejelentkeznek a számítógépjeiken feltelepített kliens programjuk segítségével. A szoftver egy TLS gRPC csatorna segítségével



2.2. ábra. Use Case

kommunikál a szerverrel. A szerveren lévő applikációhoz a rendszergazda hozzá férhet, ahol beállításokat és mentéseket készíthet. Csak a szerver képes kommunikálni a MySQL adatbázis kezelő rendszerrel.
A rendszerrel szemben támasztott általános követelmények:

- A rendszer funkcióit csak bejelentkezés után használhatják a felhasználók
- Adattárolás MySQL adatbázison
- A dokumentumok a szerveren kerüljenek eltárolásra időbélyegzővel a nevében
- A szerver és a kliens Windows platformra legyen tervezve
- TLS kapcsolat a szerver és a kliens között

2.8. Az alkalmazással szemben támasztott funkcionális követelmény

Felhasználókezelés o Admin o User Törzsadatok kezelése: Partner Partner ügyintéző Ügyintéző Évek Jelleg Csoport Telephely Iktatás: Új iktatás felvitele. A felhasználó által elkészített új iktatások megjelenítése, illetve ezekben való módosítási lehetőség. Keresés: Keresés paraméter alapján: Iktatószám Ügyintéző Tárgy Partner Jelleg Csoport Hivatkozási szám Szűrés év és irány alapján. Megjelenített iktatások számának korlátozása. Dokumentum kezelés: Feltöltés, Megnyitás, Törlés Adatbázis műveletek: Mentés és helyének megadása. Naplozási műveletek: Naplózás helyének módosítása Naplózási szint beállításának lehetősége

2.9. Funkcionális specifikáció

2.9.1. Felhasználók számára elérhető funkciók

1. **Bejelentkezés:**

   A bejelentkezési oldalon található két szövegdoboz felhasználónév és jelszó bevitelére, valamint két gomb a bejelentkezés és kilépés. A bejelentkezés gomb a szövegdobozok adatbevitele során elérhetővé válik és be lehet jelentkezni a rendszerbe. Ha az adatok hibásak voltak arról értesíti a felhasználót. A felhasználó nevének a mentésére is lehetőség, amit egy jelölőnégyzet bepipálásával lehet jelezni a szoftvernek. Sikeres bejelentkezés esetén a felhasználót a Főoldalra irányítjuk, ott nincs különösebb funkció csak a napi ige van megjelenítve. Kilépés gomb bezárja a programot.

2. **Törzsadatok:**

   Az a menüpont, ahol lehetőségünk van az iktatáshoz szükséges adatok felvitelére. Alapértelmezetten az összes törzsadat letöltődik a szerverről, ezeket jelenítjük meg több lista formájában és ezek alatt jelenik meg a hozzáadás, törlés és módosítás gomb. A hozzáadás lehetőséget ad új adat felvitelére, a módosítás a kijelölt törzs javítását teszi lehetővé, a törléssel pedig eltávolíthatjuk a nem használt törzsadatot.
   Felhasználóként a következő adatok manipulációjára van jogosultságunk:

   Ügyintéző

   Partner

   Partnerügyintéző

   Jelleg

3. **Iktatás:**

   Az erre a célra kialakított menüpontban van lehetőség az iktatásra, ahol vannak kötelezően kitöltendő és opcionális mezők. A felhasználó csak a hozzárendelt telephelyekre iktathat.

   Kötelező mezők:

   - Telephely: Ez egy legördülő lista, ahol a felhasználóhoz rendelt telephelyek választhatók ki.
   - Irány: Legördülő listaelem, két választási lehetőség bejövő és kimenő.
   - Tárgy: Ez egy szöveg beviteli mező, aminek a maximális mérete 100 karakter hosszú.
   - Jelleg: Legördülő elem, a telephelyhez tartozó jellegeket listázza ki.
   - Csoport: Legördülő elem, a telephelyhez tartozó csoportokat listázza ki.

- Partner: Legördülő elem, a telephelyhez tartozó partnereket listázza ki.
- Érkezett dátum: Ez egy dátumválasztó, a dokumentum beérkezési idejét vihetjük fel.
- Határidő dátum: Ez egy dátumválasztó, ha van a dokumentumnak határideje, akkor felvisszük, amúgy az érkezett dátumot állítjuk be neki.

Nem kötelező mezők:

- Hivatkozási szám: Szöveges mező aminek a maximális mérete XXX karakter.
- Partnerügyintéző: Legördülő elem, a törzsből kinyert partner által meghatározott adatokat tölti be.
- Megjegyzés: Ez egy szöveges mező. Az iktatáshoz egy tetszőleges hosszúságú szöveget hozzá lehet adni.

4. **Iktatás módosítása:** Minden felhasználó tud az iktatáson módosítani, de csak azokat az adatokat, amik magát az iktatószámot nem befolyásolják.

5. **Dokumentum kezelés:**

   Itt megjelennek a hozzáadott állományok, ezeket egy listában mutatjuk meg a felhasználó számára, ahol feltüntetjük a fájlok neveit, méreteit és kiterjesztésüket, illetve még itt jelenik meg a törlési lehetőség is. A lista alatt van a feltöltés, megnyitás és a mégse gomb.

   Feltöltés: Itt töltünk fel az iktatáshoz dokumentumokat, amiknek a kiterjesztése lehet pdf, docx, doc, xls, xlsx.

   Megnyitás: Miután kijelöljük a megnyitni kívánt dokumentumot, a megnyitás gomb elérhetővé válik és rákattintás után a program letölti a szerverről és az alapértelmezett programmal megnyitja azt.

   Törlés: Miután kijelöltük a törlésre szánt fájlt, a törlés gomb elérhetővé válik és rákkattintás után az adatbázisban a törölt változót igazra állítjuk, így az iktatáshoz feltöltött fájlok között nem fog megjelenni, viszont a szerveren elérhető marad.

6. **Keresés:**

   Az irány és az év kiválasztása után, a szerverről letöltődnek az adatok és listában megjelenítjük őket. A keresési sávba beírt és az ott lévő legördülő listából kiválasztott keresési feltétellel szűrhetünk az adatok között. Lehetőség az adatok módosítására illetve dokumentumok kezelésére.

7. **Kijelentkezés:**

   A kijelentkezés gomb a menüsorban található. A felhasználót kijelentkezteti a szerverről és újra megjeleníti a bejelentkezési felületet.

2.9.2. Adminok számára elérhető funkciók

A felhasználók által használt funkciók ugyanúgy az adminok által is elérhetőek, csak azokat soroljuk fel, amiben a két felhasználó különbözik.

1. **Regisztrálás:**

   A regisztrálás a rendszergazda vagy admin által történik. Az első felhasználókat csak a rendszergazda képes létrehozni.

2. **Iktatás törlése:**

   Az iktatás kétszeri kattintásával a módosítások között találhatjuk meg a törlés gombot. A gomb megnyomásával töröljük az iktatást, ha az iktatás előzmény volt, akkor a hozzá csatoltak is törlődnek.

3. **Törzsadatok:**

   Csoport és telephely adatok módosítására azért nincs lehetősége a felhasználónak, mert az intézményi ügyviteli szabályzat által vannak korlátozva.

   Felhasználó:

   Hozzáadása: A hozzáadásnál meg kell adnunk a felhasználó teljes nevét, felhasználónevét, jelszavát, jogosultsági szintjét, illetve, hogy melyik telephelyekhez tartozhat. A telephelyek kiválasztására egy legördülő menü ad segítséget, miután kiválasztottuk a telephelyet, a telephely hozzáadás gombbal tudjuk hozzárendelni a felhasználóhoz. Ezután a legördülő menüből eltűnik a kiválasztott telephely. Rontás esetén van lehetőségünk a felvitt telephely törlésére. A törölt adat visszakerül az eredeti helyére.

   Módosítása: A módosítás hasonló elven működik, mint a hozzáadás. A jelszómódosításon kívül minden adatot tudunk korrigálni. A jelszómódosításra csak a rendszergazdának van lehetősége.

   Törlése: A törléssel a felhasználót inaktívvá tesszük, minden adata megmarad, csak a rendszerbe nem fog tudni bejelentkezni. Visszaállítani csak a rendszergazda tudja.

   Telephely és Csoportok: Mind a két lehetőséghez lehet hozzáadni, módosítani és törölni. Ezeket igaz adminok végezhetik, de mind a két törzs adat

az intézményi szabályzat mondja meg, hogy mik lehetnek. Tehát tényleges adat manipuláció nagyon nem fog történni itt.

4. **Év zárás/nyitás:**

   Egy gomb ad lehetőséget az évek nyitására és zárására. Megnyomása után az aktuális évet lezárjuk és egy új évet nyitunk. Ezáltal a számozás előröl kezdődik és csak az új évre tudunk iktatni.

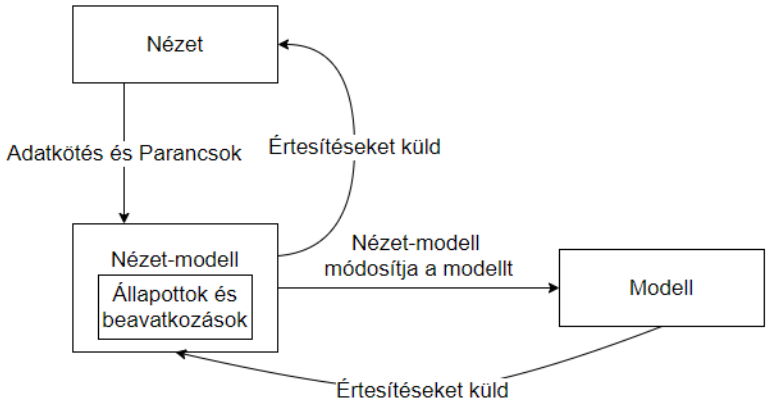
3. fejezet

Felhasznált technológiák/eszközök

3.1. MVVM

3.1.1. MVVM bemutatása

Az MVVM nem mást jelent, mint Model, View és ViewModel ami egy architektúrális tervezési minta. Ezt a mintát először a Smalltalk összefüggésében írták le a Xeroxnál, 1979-ben. Ez a minta elválasztja a GUI (Graphical user interface)-t az üzleti logikától. Kliens oldalon szokták használni. Ebben a mintában, a modell nem jelent mást, mint az adat, amit meg fogunk jeleníteni a nézetben a nézet-modellnek a segítségével. Az MVVM-mel elő jön a tisztaság fogalma, ami a sok kódra utal a code-bihend-ban. Azaz ennek a tervezési mintának a használatával nem tárolunk kódot a xaml fájl mögött. A



3.1. ábra. MVVM működése

nézet-modell úgy viselkedik, mint egy közvetítő a nézet és a modell között, és felelős a nézet logikájának a kezeléséért is. Általában, a nézet-modell interakcióba lép a modellel azáltal, hogy meghív egy eljárást a modell osztályban. Ez után a nézet-modell biztosítja az adatot a nézet számára olyan formában, hogy a nézet azt könnyedén tudja használni. A nézet-modell az adatokat a modellből szedi ki és aztán azokat elérhetővé teszi a nézet

számára, előfordulhat, hogy ezeket az adatokat egyszerűbbé kell konvertálni, hogy a nézet kezelni tudja őket. A nézet-modellben találhatóak meg azok az eljárások, amik a nézetben használt XAML elemek eseményei váltanak ki. Például ha a felhasználó rá klikkel egy gombra a UI az meghív egy metódust a nézet-modellben. A nézet-modellnek még olyan feladata van, hogy a logikai állapotokat definiálja.

A nézet és a nézet-modell, metódusok hívások, property-k, események, üzeneteken és adatkötésen keresztül kommunikálnak egymással. Az adatkötéshez a nézetünk adat kontextusához hozzáadjuk nézet-modellünket. Így a nézet-modell mezői, property-jei elérhetővé válnak a XAML elemeinek számára, ezáltal tudjuk megjeleníteni, frissíteni az adatokat a nézetben és befolyásolni a nézet-modell adatait. A nézet-modell nem csak a modell adatait tartalmazza és jeleníti azt meg a nézeten, hanem más propertiket is, mint állapot információ.

A nézet-modell felelős teljesen a modellért ebben az estben. De szerencsére ezeknek az adatoknak a kezelésére több lehetősége is van:

- A nézet-modell mutathat a modellre közvetlenül vagy propertyken keresztül
- A nézet-modell tartalmazhat interfészeket szolgáltatásokhoz, konfigurációs adatokhoz, amellyel fel tudja tölteni a propertijeit adatokkal.

3.1.2. MVVM keretrendszer

Caliburn Micro bemutatása

Ez egy kicsi, de mégis erőteljes keretrendszer, amit arra terveztek, hogy megkönnyítse a programok fejlesztését minden XAML platformon. Erős MV* minta támogatottsága segít abban, hogy az applikációinkat gyorsan, kód minőség vagy tesztelhetőség feláldozása nélkül fejleszthessük. Fejlesztők: Nigel Sampson, Rob Eisenberg és Thomas Ibel. Mivel ez is egy nyílt forráskódú projekt ezért sokan mások is hozzájárultak a fejlesztéshez. A Caliburn.micro is minden sok más keretrendszer is név konvenciót alkalmaz. Ezáltal találja meg a nézet-modell a hozzá tartozó nézetet. A keretrendszernek egyetlen függősége van és az a System.Windows.Interactivity amit igazából mindenki használ amikor WPF alkalmazást fejleszt. 27.000 kód sort tartalmaz a keretrendszer. Legfontosabb tulajdonságai:

- **Action Messages:** Az Action működése megengedi számunkra, hogy össze kössük a UI triggereit mint például gomb nyomása a nézet-modellben található metódussal. Ez a mechanizmus lehetőséget nyújt paraméterek továbbítására is a metódusok számára. A paraméter lehet egy másik FrameworkElement értéke vagy más speciális érték, mint a DataContext vagy EventArgs. Minden paraméteren elvégez egy automatikusan típus konverziót, ami a metódus szignatúrája határozz meg. Ezek a meghívások támogatják még egy védő mechanizmus is „CanExecute”-t. Ha van a triggerelt action

nevéhez hasonló metódus vagy property ami Can szóval van kiegészítve az elején akkor az blokkolhatja „disable” vagy engedélyezheti „enable” azt az elemet a UI-ban. A Caliburn.Micro ActionMassages implementációja a System.Windows.Interactivity-re épül. Ezáltal lehetőség nyílik, hogy ezt a funkciót bármilyen TriggerBase alapú triggerre használhassuk.

– **Action Conventions:** A konvenciók a x:Name-re épülnek. A nézetben lévő elemet és a nézet-modellben található megegyező nevű metódusra automatikusan készít egy EventTriggert az elem által alapértelmezetten definiált eseményre, és hozzárendel egy ActionMessage-t ami metódusra fog mutatni. Ha van egy metódusunk amit „Login”-nak hívunk és van egy gombunk UI-on aminek a neve ugyanúgy „Login” akkor arra a keretrendszer automatikusan összeköti. Így amikor a gombra rá kattintunk a login metódus fog meghívódni. Továbbá figyeli a metódus szignatúráját és a szerint építi fel az ActionMassage paramétereit. Ez csak egy lehetőség, amit bármikor kikapcsolhatunk, vagy tetszés szerint módosíthatunk. Ezeken kívül módosíthatunk, hozzáadhatunk új konvenciókat különböző elemkehez. Például, tudunk olyat is csinálni, amivel feliratkozunk a gomb MouseMove eseményére.

– **Binding Conventions:** A Caliburn.Micro támogatja a konvenció-alapú adatkötést is. Ez ugyanúgy x:Name segítségével is működik. Hogyha van egy property a nézet-modellben ugyan olyan névvel, mint az elem neve akkor azt megpróbálja összekapcsolni. Mikor egy összekötés bekövetkezik akkor a keretrendszer több lépésen keresztül építi a kapcsolatot. Ezek közül mind testre szabható, mint például az adatkötés módja azaz BindingMode vagy StringFormat, ValueConverter stb. . .

– **Screen, Conductor és Collection:** A Screen, Conductor és Collection osztály felel a nézetünk életciklusáért azaz a helyes indításért, leállításért és leállítás megszakításáért az applikációnkban. A screen-t usercontroll nézet-modellkén használják. Ezek az építő elemei a megjelenítésnek a caliburn.microban. A conductor és Collection is a Screen osztály leszármazottjai. A Conductor is lehet használni usercontrolloknál de ezt már ablakoknál vagy oldalaknál alkalmazzák javarészt, lényege, hogy képes kezelni egy Screen típusú osztályt, amit meg tud jeleníteni. A Collectionek két fajtája van. A Collection.AllActive és a Collection.OneActive. Mindegyik több screen befogadására képes, de az egyik egyszerre csak egy screent jelenít, meg míg a másik akár egyszerre az összeset is.

– **Event Aggregator:** Az aggregátor egy pub/sub tervezési mintára épülő osztály. Azaz feliratkoztatjuk az egyik osztályunkat és az üzeneteket fog nekünk küldeni attól függően, hogy a regisztrált osztály milyen IHandle<T> interfészeket implementált. Az értesítés az UI szálon történik. Támogatja a polimorfizmust is.

– **View Locator:** Minden nézet-modellhez a programunkban van egy alap stratégia a hozzá tartozó nézet megtalálásához. Ezt név konvenció alapján végzi el. Például ha a nézet-modellünket elnevezzük IktatogRPCClient.ViewModels.ContainerViewModenek akkor a Caliburn.Micro a IktatogRPCClient.Views.ContainerView-t fogja keresni. Egy nézetnek akár több nézet-modellje is lehet.

– **View Model Locator:** Attól függetlenül, hogy alapértelmezetten a Caliburn.Micro a nézet-modelltől indul ki alapértelmezetten ez módosítható. Azaz támogatja, hogy először a nézetet keresi és utána nézet-modellt ugyan azzal az eljárással, mintha ennek az ellenkezője lenne.

– **Window Manager:** Ez a szolgáltatás is jelzi, hogy ez a keretrendszer nézett-modell központú. Átadunk a számára egy példányt abból a nézet-modellből, amit megszeretnénk jeleníteni, ő megkeresi a hozzátartozó nézetet, beállítja az összes konvenciót, amit beállítottunk rajta és megjeleníti az ablakot.

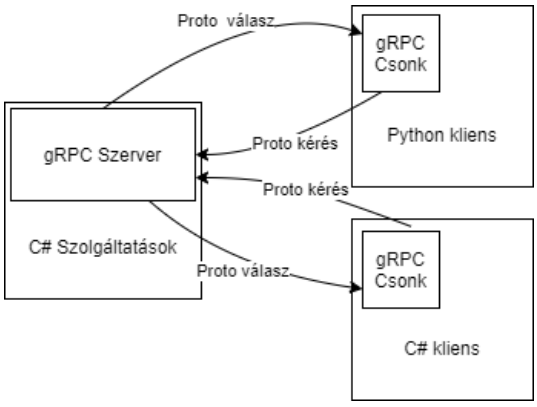
– **PropertyChangedBase and BindableCollection:** A keretrendszerben lévő implementáció lehetővé teszi számunkra, a string és lambda kifejezés alapú változás értesítéseket. Megbizonyosodik róla, hogy minden esemény ami a propertyhez kapcsolódik a UI szálon történjen. A BindableCollection az ObservableCollection leszármazottja, azaz egy egyszerű lista azonban itt is ügyel arra, hogy minden eseménye a UI szálon történjen.

– **Bootstrapper:** Ennek az osztálynak a segítségével tudjuk a szoftverünket elindítani és futtatni. Nem kell, mást tenni hozzá csak létrehozunk egy saját osztályt, ami a BootstrapperBase osztály leszármazottja és hozzáadjuk azt a program ResourceDictonarybe. Innen tudjuk begyűjteni még az indítási paramétereket is.

3.2. gRPC és a Protobuffer

3.2.1. gRPC bemutatása

A gRPC rekurzív mozaik szó. A jelentése gRPC Remote Procedure Call. A google fejlesztette 2015-ben. Nyílt forráskódú. Segítségével közvetlenül meghívhatunk metódusokat egy távoli gépen, mintha az lokális objektum lenne. Könnyedén létrehozhatunk vele osztott alkalmazásokat, szolgáltatásokat. Két fő részből áll. A gRPC protokollból és a protobufferből. A többi RPC szolgáltatáshoz hasonlóan itt is előre kell definiálni a szolgáltatásokat. Ezen szolgáltatások metódusait, amelyeket a későbbiek folyamán meg lehet hívni a paraméterek megadásával. A gRPC platform független. Támogatott nyelvek: c#, java, c/c++, Dart, Go, Kotlin, Node, Objective-C, PHP, Python, Ruby.



3.2. ábra.

Egyik előnye, hogy automatikusan generálja nekünk a kliens csonkokat és a szerver osztályokat.

gRPC Protokoll

A protokoll http2 alapokon működik és ki is használja annak előnyeit. A gRPC támogat több beépített lehetőséget, amit a http2-től örökölt. Például fejléc tömörítése, folyamatos és egyetlen TCP kapcsolat, csatlakozási, megszakítási és időtúllépési szerződések a kliens és a szerver között. A protokollnak van beépített folyamatirányítás vezérlése az adatkereteken, amit ugyanúgy a http2-től örökölt. Ez nagyon hasznos annak biztosítása érdekében, hogy az kliensek tiszteletben tartsák a rendszer teljesítményét, de extra összetettséget eredményez az infrastruktúrában felmerülő problémák diagnosztizálásakor, mivel akár az ügyfél, akár a szerver beállíthatja saját folyamatirányítási értékeit. A gRPC szerver fogja kezelni a kliens hívásokat. A kliens oldalon van egy csonk, ami azoknak a metódusoknak a fejlécét tartalmazza, amik a szerveren is megtalálhatók. Ezek segítségével fogjuk meghívni a szerver oldalon a service által definiált metódusokat.

3.2.2. Protobuffer bemutatása

A protocol buffer egy nyelvi és platform semleges protokoll, ami az adatok szerializációjáért felel. A Protocol buffer tölti be a szerződés szerepét a kliens és a szerver között. Sokkal egyszerűbb, hatékonyabb, mint egy XML struktúra és könnyebben olvasható is. Két részből áll az egyik a Message a másik a Service amiben az RPC-k találhatóak.

Message

A message egy üzenet, ami az rpc híváskor adható meg paraméterként illetve ilyen üzeneteket tud visszaküldeni a szerver. Itt lehet osztályokat kialakítani. Továbbá az rpc hívásoknál nem lehet üres a metódus ezért létre kell hozni egy olyan Message-t

is aminek nincs semmilyen mezője és ezt kell megadni paraméternek. A messagekben a skalár típusok találhatóak meg. Pl.: int32,int64,double,string,bool stb. Egy message három részből áll. Az első megmutatja az adat típusát, amit már az előbb említettek közül választhatunk ki vagy lehet egy újabb message is. A második rész a neve a mezőnek. A harmadik rész egy egyedi szám. Ez a szám segít meghatározni a mezőinket, amikor az üzenetünk bináris formátumban van.

Service

A service nem más, mint azoknak a RPC eljárásoknak gyűjtő osztálya, amit használni szeretnénk. Egy gRPC szerver több servicet is ki tud szolgálni egyszerre egy adott socketen. Alapból a compiler generál egy absztrakt osztályt és egy hozzá tartozó típus biztos csonk implementációt. A csonk fogja továbbítani a hívásokat egy gRPC csatornán keresztül.

A gRPC RPC típusai

Unary: Alapvetően ezek egyszerű szinkron kérések, amiket a kliens szoftver végez a gRPC szerver felé végzünk. Egy ilyen kérés blokkolja addig a szálat amíg a szerver nem válaszol.

Client streaming: A szerver oldal több üzenetet fogad a kliens oldal felől, miután a kliens befejezte az adatok küldést a szerver egy válasszal zárja a kapcsolatot.

Server Streaming: Ez a kliens streamingnek a fordítottja, de itt ugyanúgy a kliens kezdi a kommunikációt. Küld egy üzenetet és erre a szerver több üzenettel válaszol.

Bidirectional streaming: A kliens és a szerver között egy állandó kapcsolat épül ki. Egymástól függetlenül tudnak egymásnak adatokat küldeni.

Proto fájlból DLL

Miután megírtuk a .proto fájlunkba a serviceket az rpcket és a messageket akkor van lehetőségünk őket egy protofordító segítségével olyan nyelvi kóddá fordítani, azok közül a nyelvek közül választhatunk, amiket a gRPC támogat. Ezek a fájlok tartalmazni fognak a nyelvhez illeszkedő objektumokat a messagek számára. Ezek az objektumoknak az adatait getteren és setteren keresztül fogjuk elérni. Illetve létre hoz még egy osztályt is a service-ek számára. Ha c# nyelvet választottuk, akkor miután legeneráltattuk ezeket a fájlokat be kell tennünk a visual studioba után buildeljük és az számunkra létre hozza a DDL fájlokat. Ezeket az állományokat már könnyedén hozzáadhatjuk a szerver és a kliens oldali projektünkhöz.

4. fejezet

Implementálás

4.1. Kliens implementálása

Ebben a részben be fogom mutatni a kliens oldali programom részeit. Ezeket részeket a menü felépítésének sorrendjében fogom ismertetni. A projekt számos nézetet és nézet-modellt tartalmaz ezért csak főbb összetevőit és problémásabb részeket fogom kiemelni. A legfontosabb elemei a bejelentkezés, konténer, főoldal, iktatás, keresés, törzsek kezelése. Ezeknek a nézetekhez tartozó nézet-modellek egyes metódusait és property-jeit, és a ServerHelperSingleton osztályt fogom bemutatni röviden. A kliensre a Caliburn.Micro elérhető a nuGet package manageren keresztül amit így egy kattintással telepíthetünk.

Indítás/alapbeállítás:

Az indításhoz való legelső konfigurációs lépés nem más volt, mint az applikáció App.xml fájlja én sem tettem mást, mint a dokumentációkban leírtak alapján hozzáadtam az indító fájlt illetve kivettem a StartupUri részt az application részből.

```
1  <Application.Resources>
2          <ResourceDictionary>
3                  <ResourceDictionary.MergedDictionaries>
4                          <ResourceDictionary>
5                                  <local:Bootstrapper x:Key="Bootstrapper"
                                       ></local:Bootstrapper>
6                          </ResourceDictionary>
7                          <ResourceDictionary Source="style.xaml"/>
8                  </ResourceDictionary.MergedDictionaries>
9          </ResourceDictionary>
10 </Application.Resources>
```

Mint látható a ResourceDictionary-hoz egy MergedDictionary-t adtam hozzá. Az egyik dictionary tartalmazza az indítani kívánt osztályt, aminek a BootstrapperBase osztálynak az ősének kell lennie. A másik tartalmazza a szoftver formázási stílusait így XAML fájlba nem kell közvetlenül beleírnom ezeket a megjelenítési beállításokat és így azok újra felhasználhatóvá válnak és átláthatóbbá teszi a nézet kódját.

```
public class Bootstrapper:BootstrapperBase
{
        public Bootstrapper()
        {
                Initialize();
        }
        protected override void OnStartup(object sender, StartupEventArgs
            e)
        {
                LogHelper.Initialize();
                if (e.Args.Contains("-d")) LogHelper.SetLoglevel(
                    LogEventLevel.Debug);
                DisplayRootViewFor<LoginViewModel>();
        }
}
```

Létrehoztunk egy Bootstrapper osztályt a BootstrapperBase osztály leszármazottjaként. Az osztálynak nyílvánosnak kell lennie, hogy az App.xaml-ben is látható legyen. Felülírtuk az OnStartup metódusát annak az érdekébe, hogy a saját általunk írt ViewModelt tudjuk elindítani. Ha sorról sorra megyünk a metóduson belül, akkor látszik, hogy először a LogHelper osztályt alapállapotba állítjuk. A logoláshoz a Serilog-t használom. Ha van –d paramétert indításkor akkor a logolást debug módban fogjuk beállítani így szinte mindenről fogunk tudni, hogy a felhasználó mit, mikor csinált az adott classban. Alapértelmezetten Warning-ra van állítva, azaz csak az olyan hibákat írjuk a logba ami kivételt okozott a programban. DisplayRootViewFor<T>() metódus segítségével el indítjuk a Loginunkat ahol is a T helyére beírtuk az indítani kívánt nézet-modellt. Ezután a korábbiakban leírt módon meg keresi a hozzá nézetet és megjeleníti azt.

Bejelentkezés:

Ebben a nézet-modellben összesen három property található. Ezekbe tárolom el a felhasználó adatait illetve, egy bool propertyt ami jelzi, hogy a felhasználó szeretné-e, hogy elmentsük a jelszavát. Ha szeretné a jelölő kattintásával módosítja a nézet-modellben található property-t és bejelentkezéskor az adatot a windows registrybe mentem. Ennek a segítségére megtalálható három konstans, ami az adat elérési útját tartalmazza. Ezen kívül megtalálható egy privát mező is ami a UserHelperSingleton osztály, mivel a Login lehetőség ebben az osztályban érhető el. Több metódus is a rendelkezésünkre áll ezek közül a legfontosabb a szerverhez való csatlakozásra használt.

```
async Task<bool> ConnectToServerAndLogin()
{
        CheckUsername();
        CheckPassword();
        Log.Debug("{Class} Csatlakozas a szerverhez", GetType());
```

```
6           bool success = await userHelper.Login(GetDataFromLoginTextBoxes()
                );
7           if(success)Log.Debug("{Class} Sikeres csatlakozas", GetType());
8           else Log.Debug("{Class} Sikertelen csatlakozas", GetType());
9           if (SaveUsernameIsChecked) Registry.SetValue(keyName, "
                Felhasznalonev", UsernameBox);
10          else Registry.SetValue(keyName, "Felhasznalonev", "");
11          return success;
12  }
```

Ez egy async metódus annak érdekében, hogy ne blokkoljuk a UI threadet a csatlakozás közben így a kis töltést jelző indikátorunk nyugodtan futhat rajta. Az elején ellenőrizzük a felhasználó által beírt adatokat, ha hiba van az adat kitöltésben például nem írt be jelszót vagy túl rövid alapból, akkor egy InvalidUserNameExceptiont dobunk és ezzel megszakítjuk a metódus működését. Ha minden rendben van, elküldjük a beírt adatokat a szerver felé. Sikeres bejelentkezéskor megnyitjuk a konténer nézetet és abban egyből a főoldalt, illetve el tároljuk a válaszban kapott tokent. Hibás jelszó esetén a gRPC szerver kivált egy kivételt, amit eljuttat a kliens oldalra és ott jelezzük a felhasználónak azt.

Konténer:

Ebben a nézetben található meg a menüsor, illetve itt fog változni az egész program. Azaz amikor egy menü gombra rákattintunk, ami nem kilépés vagy kijelentkezés akkor az adott usercontroller betöltésre kerül. Ez egy conductor osztály amit a caliburn.micro keretrendszere nyújt. Így mindig lehet egy aktív screenem. Ennek a segítségére létre hoztam egy külön osztályt amit SceneManager-nek neveztem el. Ennek az osztálynak van egy CreateScene függvénye aminek egy paramétere a Scenes enumok közül választható. Visszatérési érteke egy screen osztály béli objektum. A Scenes enumok nem mást tartalmaznak, mint a megjeleníthető nézetek listáját.

```
1   private void ChangeScene(Scenes scene) {
2           Log.Debug("{Class} Oldal valtasa {Scene}", GetType(),scene);
3           ActivateItem(SceneManager.CreateScene(scene));
4   }
```

Ezt a metódust a gombokhoz hozzárendelt metódusok hívják meg. Így egy helyen történik a „jelenet” váltás. Az ActivateItem a Conductor osztály egy metódusa aminek a bemenő paramétere egy Screen objektum. Ez a metódus fogja a nézetben található contentcontrol elembe beletenni a meghívott usercontrollt. Ezt is név konvenció alapján köti össze a telepített keretrendszer. A contentcontrollnak „ActiveItem”-nek kell lennie az x:Name mezőjében.

Főoldal:

A főoldalon egy üdvözlő üzenet jelenik meg, illetve a vágyálom rendszerben is megfogalmazott napi ige. Erre a főoldal nézet-modellje az ige modellt használja. Az ige

osztály három propertyből áll és két mezőből. A LetoltottIgeDatuma jelzi, hogy mikor volt letöltve az ige. Ennek van egy alapértelmezett dátuma, ami biztos kisebb, mint az aktuális nap dátuma. Van egy property aminek csak getterje van, ez a property szolgáltatja a címet, ami magába foglalja az aktuális nap dátumát. Az utolsó pedig maga a napi ige tárolására szolgál. A főoldalon ezt a modellt statikusként állítottam be, annak érdekében, hogy a nézet-modell ne végezze el az ige adatok letöltését és formázást, csak ha szükséges. A letöltéshez a következő metódust használom:

```
1  private async void DownloadIge()
2  {
3          try
4          {
5                  string result = "";
6                  using (WebClient client = new WebClient())
7                  {
8                          client.Encoding = Encoding.UTF8;
9                          result = await client.DownloadStringTaskAsync("
                               https://napiige.lutheran.hu/igek.php");
10                 }
11                 _ige.NapiIge = BuildDailyIge(result);
12                 _ige.LetoltottIgeDatuma = DateTime.Today.Date;
13                 NotifyOfPropertyChange(()=>NapiIge);
14         }
15         catch (WebException e)
16         {
17                 InformationBox.ShowError(e);
18         }
19 }
```

Az aszinkronitás itt is megvan azért, hogy a UI szálat ne foglaljuk le. Először létrehozok egy WebClient ami a System.Net névtérből származik. A kliensnek beállítom a kódolását. Ezután a fenntartótól kapott API –n keresztül leszedem a napi igét. Ezek után formázom azt reguláris kifejezések segítségével és egy kis trimmeléssel mivel ezek forráskódként kerülnek letöltésre, majd beállítom a letöltésnek a dátumát és szólok a UI-nak, hogy frissítse a NapiIge-vel összekötött elemét. Hiba esetén jelzem azt a felhasználó számára.

Iktatás:

Az iktatás nézetében, megtalálható az összes kötelező törzs adat kiválasztási lehetősége. Ezeket legördülő menü formájában találja meg a felhasználó a felületen. Ezek mindegyikéhez a nézet-modellben tartozik egy név azonos property, illetve a kijelölt elemek számára is. A nézet-modell-t az IkonyvHandlerből származtatom, ami tartalmazz két IHandle interfészt és a Screen osztály leszármazottja. Ezt a segéd osztályt azért hoztam létre, hogy csoportosítsam egy helyre azokat a tulajdonságokat egy nézetnek, amit muszáj kezelnie, ha tartalmaz olyan DataGridet amiben iktatásokat jelenít meg.

Három metódus tartalmazz ebből kettő absztrakt, hogy azt a gyermekosztály majd megvalósítsa. Az iktatókönyv módosítására való metódus viszont mindenhol ugyan az ezért azt már itt implementáltam.

```
1 public void ModifyIkonyv() {
2         if (SelectedIkonyv == null) return;
3         WindowManager windowManager = new WindowManager();
4         Screen screen = new PopUpViewModel(new ModifyIkonyvViewModel(
              _selectedIkonyv));
5         windowManager.ShowDialog(screen);
6 }
```

Ebben a metódusban található egy WindowManager ami az új ablak megnyitásáért felelős a Caliburn.Micro keretrendszerben. A ShowDialog függvénynek a bemeneti paramétere egy PopUpViewModel lesz, aminek át adunk egy másik nézet-modellt ez pedig az iktatás módosítására használt objektum. A PopUp nézetett azért hoztam létre, hogy az ablak mögötti rész el sötétítése egy helyen történjen, ne kelljen folyamatosan újra implementálnom, a hibákon kívül minden felugró ablak ezen a nézet-modellen keresztül fog megjelenni.

Az iktatás nézet-modell konstruktorában először csak a telephelyeket szedem, le az adatbázisból majd beállítom az első lehetséges telephelyet. Ennek hatására megindul a telephelyhez tartozó adatoknak is a letöltése.

```
1 private async void LoadData()
2 {
3         Log.Debug("{Class} Adatok betoltese a szerverrol.", GetType());
4         SelectedIrany = Iranyok.First();
5         eventAggregator.Subscribe(this);
6         AvailableTelephelyek = await serverHelper.GetTelephelyekAsync();
7         if (AvailableTelephelyek.Count > 0) SelectedTelephely =
              AvailableTelephelyek.First();
8         Log.Debug("{Class} Sikeres adat letoltes.", GetType());
9 }
```

Az irányok belevannak „égetve” az osztályba mert, ezek nem fognak változni. Alapértelmezetten kijelöli a „bejövő” részt tehát erre fog iktatni a felhasználó, ha ezt nem módosítja. A konstruktorban még fel is iratkozok a Caliburn.Micro által nyújtott EventAggregatorra. Az eventAggregator segít abban, hogy az osztály értesítéseket kapjon azokról az eseményekről, amikre az IHandle interfészek által feliratkozott Az IHandle implementálni kell, mivel a KonyvHandler osztályban nincsenek.

A Handlerek implementációja:

```
1 public override void Handle(RemovedItem message)
2 {
3         if (message.Item is Ikonyv)
4         {
```

```
5                   _recentlyAddedIkonyvek.Remove(message.Item as Ikonyv);
6                   NotifyOfPropertyChange(() => RecentlyAddedIkonyvek);
7           }
8   }
```

A RemovedItem handler arra szolgál, ha törölné a felhasználó az éppen felöltött iktatást, akkor az a statikus Iktatókönyvek közül kitörölje.

```
1   public override void Handle(Ikonyv message)
2   {
3           for (int i = 0; i < RecentlyAddedIkonyvek.Count; i++)
4           {
5                   if (RecentlyAddedIkonyvek[i].Id == message.Id)
6           {
7           RecentlyAddedIkonyvek.RemoveAt(i);
8           RecentlyAddedIkonyvek.Add(message);
9   }
```

A módosított iktatásokat kapja el. Ha egy módosítás bekövetkezik, akkor azt kitöröli a nem rég hozzáadott iktatókönyvek közül és hozzáadja az új módosított objektumot.

Ha a felhasználó iktatni szeretne és elkezdi ki tölteni a mezőket, azt folyamatosan ellenőrizzük, ha a bevitt adat elegendő és az összes kötelező mezőt kitöltötte, akkor az iktatási gomb, ami alapértelmezetten kikapcsolt állapotba van, bekapcsoltra állítjuk. Ennek a funkciónak a megkönnyítésére is van lehetőség Caliburn.Micro-ban amit ugyanúgy névkonvenció alapján összetudja kötni a logikai propertyt a XAML elemmel. A gombot el neveztük IktatButton-nak így a publikus property amit létre kell hoznunk az a CanIktatButton. Ezen a property-n belül csak vissza adom az én saját private logikai függvényem értékét ahol történik az igazi ellenőrzés.

```
1   private bool FormValidation() {
2           bool IsValid = true;
3           if (IsTorzsDataInFormNull()) IsValid = false;
4           else if (string.IsNullOrWhiteSpace(Targy)) IsValid = false;
5           else if (Targy.Length > 100) IsValid = false;
6           else if (DateTime.Parse(HatidoDatum) < DateTime.Parse(
                ErkezettDatum)) IsValid = false;
7           else if (ValaszIsChecked && SelectedIktSzam == null) IsValid =
                false;
8           else if (Szoveg.Length > 500) IsValid = false;
9           else if (Hivatkozasiszam.Length > 50) IsValid = false;
10          return IsValid;
11  }
```

Az függvény folyamán ellenőrzöm a szöveges mezők számának a hosszát, hogy az nem haladta-e meg az adatbázisban foglalt oszlop maximális karakter számát. Továbbá ellenőrzöm még azt, hogy a határidő dátuma ne legyen kisebb, mint az érkeztetés dátuma

és még azt is, hogy minden kötelező mező ki lett-e töltve. Ha ez sikeres, akkor az iktatás gombra rá tudunk kattintani, ahol nem történik más, mint létrehoz egy új iktatási objektumot, ami a ServerHelperSingleton által szolgáltatott AddIktatas függvényét meghívjuk.

A függvény bemeneti paramétere egy Ikonyv típusú objektum a kimeneti paramétere egy RovidIkonyv ami csak az Id-t és az iktatószámot fogja tartalmazni. Ez a gRPC által generált kliens csonkot fogja használni arra, hogy a szerverre elmentse az adatokat.

```
1  public async Task<RovidIkonyv> AddIktatas(Ikonyv newIkonyv)
2  {
3          RovidIkonyv rovidIkonyv = new RovidIkonyv() { Id = 0 };
4          try
5          {
6                  rovidIkonyv = await Client().AddIktatasAsync(newIkonyv,
                       GetCallOption());
7          }
8          catch (RpcException ex)
9          {
10                 InformationBox.ShowError(ex);
11         }
12         catch (Exception e)
13         {
14                 InformationBox.ShowError(e);
15         }
16         return rovidIkonyv;
17 }
```

A visszatérés után megjelenítjük a felhasználónak az új iktatószámot és a kiegészített adatokkal hozzáadjuk a RecentlyAddedIkonyvek propertyhez. Ezáltal meg fog jelenni a nézetben is.

Keresés:

A keresés is gyermek osztálya a IkonyvHandler osztálynak. Első lépésként leszedem a szerverről az elérhető éveket és beállítom a lehetséges irányokat. Amint a felhasználó mind a két legördülő menüből kiválasztott egy lehetőséget megkezdődik a szerverről az iktatásoknak a letöltése. Az adatok megjelenítéséhez három BindableCollection<Ikonyv> property-t kell, amik a következők: AllIkonyv, SearchedIkonyvek, ShownIkonyvek. Az első property tartalmazza az összes iktatást, amit leszedtünk, a második tartalmazza azokat az adatok, amik megfelelnek a keresési kritériumoknak és a harmadik maga a megjelenített iktatások. A megjelenítéskor figyelembe veszem, hogy a felhasználó hány darab iktatást szeretne megjeleníteni az oldalon és csak annyit jelenítek meg. Az oldalak kezeléséhez a gombokat le kell generálni, aminek darabszámát mindig a SearchedIkonyvek található adatok száma és a megjeleníteni kívánt iktatás

oldalanként határozza meg. Az összes letöltött adat darab számát megjelenítem.

```
1  private async void SetVisibleIktatas() {
2          if (AllIkonyv.Count == 0) return;
3          await SetSearchData();
4          ShownIkonyvek.Clear();
5          if ((CurrentPage+1) * SelectedItemsPerPage > SearchedIkonyvek.
               Count)
6          {
7                  for (int i = (CurrentPage ) * SelectedItemsPerPage; i <
                       SearchedIkonyvek.Count; i++)
8                  {
9                          ShownIkonyvek.Add(SearchedIkonyvek[i]);
10                 }
11         }
12         else
13         {
14                 for (int i = CurrentPage * SelectedItemsPerPage; i <
                       CurrentPage * SelectedItemsPerPage +
                       SelectedItemsPerPage; i++)
15                 {
16                         ShownIkonyvek.Add(SearchedIkonyvek[i]);
17                 }
18         }
19         NotifyOfPropertyChange(() => ShownIkonyvek);
20         SetButtons();
21         NotifyOfPropertyChange(() => MaxItemNumber);
22 }
```

A megjelenítésnek a folyamata a következő képen zajlik. Ellenőrzöm, hogy van-e egyáltalán iktatás a listánkban, ha nincs kilépek az eljárásból. Utána megnézem, hogy a felhasználó beírt-e valami keresési feltételt, ha igen akkor annak alapján beállítom a SearchedIkonyv propertyt miután ez megtörtént megnézem, hogy kitudom-e íratni az összes talált iktatást, ha nem akkor megnézem, hogy melyik oldalon is vagyunk és az alapján iratom ki azokat. Miután ezt befejeztem szólok a nézetnek, hogy frissítse a kötéseket, beállítom a gombokat és jelzem a talált iktatások számát. **Törzs:**

A nézet-modell a Collection.AllActive gyermek osztálya. Mint azt már korábban említettem, ez az osztály abban különbözik a Conductor osztálytól, hogy több Screen eleme is lehet és egyszerre ezek közül többet is meg tud jeleníteni. Ahhoz hogy megjelenítsük ezeket a nézet-modelleket nincs más teendőnk, mint létrehozni minden egyes megjeleníteni kívánt Screen típusú osztálynak egy property-t amit a caliburn.micro össze köt a XAML-ben található contentcontroller elemekkel. Itt megnézzük, hogy a felhasználó admin-e ha igen, akkor az összes beállítási lehetőségeket megkapja. Ebben a törzs nézet-modellben történik még meg a telephelyek letöltése és publikálása, de csak miután megjelenítettük és felírattuk az összes többi nézet-modellt az eventaggre-

gatorra.

```
1  private async void GetTelephelyekAsync() {
2          Log.Debug("{Class} Telephelyek letoltese.", GetType());
3          LoaderIsVisible = true;
4          BindableCollection<Telephely> telephelyek = await
               ServerHelperSingleton.GetInstance().GetTelephelyekAsync();
5          await eventAggregator.PublishOnUIThreadAsync(telephelyek);
6          LoaderIsVisible = false;
7  }
```

Mint látható a TorzsViewModel is a ServerHelperSinglton osztályt alkalmazza az adatok letöltésére. Miután ez megtörtént az EventAggregatornak van egy metódusa, hogy PublishOnUIThreadAsync() aminek a bemeneti paraméter egy object osztály. Tehát bármilyen osztályt alkalmazhatunk ezáltal. Ezután a hívás után azok az elemek is elkezdhetik a saját adataiknak a letöltését a szerverről, amelyek implementálták az IHandle<Telephely> interfészt. Mindegyik nézet-modell implementálja azokat az IHandle<T> interfészeket, amikre szükségük van. Ebben a menüpontban van a legjobban kihasználva az EventAggregator és az ServerHelperSingleton osztálynak a tulajdonságai. Mindegyik törzset kezelő felület tartalmazza szerver elérésére létrehozott egyke osztályt, ennek hála mindegyik lekérdezés ugyan azon a csatornán fog végződni mivel ebből az osztályból csak egy van az egész program futtatása alatt.

Dokumentumkezelés:

A dokumentum kezelésre minden olyan felületnek van lehetőség amelyik implementálja a IkonyvHandler osztályt. Ugyanis az iktatáshoz tartozó dokumentumok kezelésére a módosítás közben van lehetőségünk. Amint rákattintunk kétszer egy iktatásra a DataGrid-en az egy új ablakot fog megjeleníteni ahol is található a dokumentumok gomb. Ennek a gombnak a megnyomásra jelenik, meg az a felület ahol látjuk az összes dokumentumot és ahol fel is tudunk tölteni újat. A dokumentum megnyitása úgy történik, hogy kijelöljük a megnyitni kívántat és rákattintunk a megnyitás gombra. Ennek hatására a DownloadDocument metódus fog meghívódni.

```
1  public async void DownloadDocument() {
2          try
3  {
4  LoaderIsVisible = true;
5          Document rawdata = await serverHelper.GetDocumentByIdAsync(
               SelectedDocument);
6          if (rawdata.Id != -1){
7                  Log.Debug("{Class} Sikeres letoltes", GetType(),
                       SelectedDocument);
8                  byte[] bytes = rawdata.Doc.ToByteArray();
9                  string temppath = Path.GetTempPath();
10                 string fullpath = $"{temppath}{rawdata.Name}.{rawdata.
                       Type}";
```

```
11                  Log.Debug("{Class} Adatok mentese", GetType(),
                        SelectedDocument);
12                  if (File.Exists(fullpath)){
13                          File.Delete(fullpath);
14                          File.WriteAllBytes(fullpath, bytes);
15                  }
16                  else File.WriteAllBytes(fullpath, bytes);
17                  Log.Debug("{Class} Sikeres adatmentes.", GetType(),
                        SelectedDocument);
18                  try{
19                          Log.Debug("{Class} Letoltott documentum
                                megnyitasa.", GetType(), SelectedDocument);
20                          Process.Start(fullpath);
21                  }
22                  catch (Exception e){
23                          InformationBox.ShowError(e);
24                  }
25          }
26          else{
27                  MessageBox.Show("Hiba a dokumentum letoltese kozben.");
28          }
29          LoaderIsVisible = false;
30          catch (Exception e){
31                  InformationBox.ShowError(e);
32          }
33  }
```

A függvény először vár az adat letöltésére, ha sikerült, akkor visszakapjuk annak a fontos információit és magát a dokumentumot. Ha az Id mező nem egyenlő -1 -el akkor megyünk tovább. Az adatokat berakjuk byte tömbbe, lekérjük a rendszertől az ideiglenes fájlok tárolására használta mappának az elérési útvonalát. Ha ez sikeres, akkor konkatenáljuk ezt az útvonalat a letöltött dokumentum nevével. Ha már letöltötték ezt a fájlt, akkor először kitöröljük, majd újra kiírjuk a lemezre, ha nem akkor csak kiírjuk. Ezután a Process.Start() segítségével megnyitjuk az állományunkat. Ennek a metódusnak több változata is van, de én a string paramétereset hívom meg, ami fájl vagy dokumentum elérési útvonalának kell lenni ennek hatására elindítja az alapértelmezett programmal azt.

A feltöltés gomb az UploadDocument metódust indítja el. Először megjelenítjük a c# által támogatott fájl kiválasztót, ha ott sikeresen kijelöltünk egy fájlt létrehozunk egy új Document objektumot. Itt minden információt megadunk a fájlról és beolvassuk a Doc mezőjébe a fájl adatait.

```
1  public async Task UploadDocument() {
2          Log.Debug("{Class} Feltoltes gomb megnyomva.", GetType());
3          Log.Debug("{Class} Adatok beszerzese", GetType());
```

```
4           string[] FileInfo = ChooseDataToUpload();
5           if (string.IsNullOrWhiteSpace(FileInfo[2])) return;
6           Log.Debug("{Class} Fajl feltoltes megkezdese.", GetType());
7           LoaderIsVisible = true;
8           Document document = new Document();
9           document.Name = FileInfo[0];
10          document.Type = FileInfo[1];
11          Log.Debug("{Class} ByteArray to BysteString", GetType());
12          document.Doc = ByteString.CopyFrom(GetBytesFromFile(FileInfo[2]))
                ;
13          document.IkonyvId = ikonyvid;
14          Log.Debug("{Class} Adatok feltoltese. Doc: {Name}", GetType(),
                document.Name);
15          DocumentInfo uploadedDocument = await serverHelper.
                UploadDocumentAsync(document);
16          if (uploadedDocument.Id != -1)
17          {
18                  Log.Debug("{Class} Sikeres feltoltese. ", GetType());
19                  IkonyvDocuments.Add(uploadedDocument);
20                  NotifyOfPropertyChange(() => IkonyvDocuments);
21                  ModificationHappend = true;
22          }
23          else {
24                  Log.Debug("{Class} Sikertelen feltoltese. ", GetType());
25                  MessageBox.Show("Sikertelen feltoltes.");
26          }
27          LoaderIsVisible = false;
28  }
```

Sikeres beolvasás után elindítjuk a feltöltést, ellenőrizzük annak sikerességét. Hozzáadjuk a dokumentumot a jelenlegiekhez és értesítem a nézetet a változásokról.

SeverHelperSingleton:

Ez az osztály a legfontosabb a szerver eléréséhez, ugyanis ő használja a kliens csonkokat. Ő az egyke osztályok azon fajtáját reprezentálja, amit úgy hívunk, hogy „mohó”. A mohó osztály annyiban különbözik a „lustától” hogy alapból szálbiztos, de erőforrásait egyből inicializálja. A csonkok használatán kívül más feladatai is vannak. Az egyik a gRPC csatorna létrehozása és annak lezárása a másik, hogy a hitelesítési adatokat mindig elküldje a http2 fejlécében a szerver számára.

```
1   private void CreateNewChannel() {
2           string hostname = ConfigurationManager.AppSettings["Hostname"];
3           string hostport = ConfigurationManager.AppSettings["Port"];
4           string csatinfo = $"{hostname}:{hostport}";
5           var servercert = File.ReadAllText("cert/server.crt");
6           SslCredentials creds = new SslCredentials(servercert);
7           channel = new Channel(csatinfo, creds, new[] {
```

```
8                   new ChannelOption("grpc.keepalive_permit_without_calls",
                        1),
9                   new ChannelOption("grpc.http2.max_pings_without_data",0),
10                  new ChannelOption("grpc.keepalive_timeout_ms",50),
11                  new ChannelOption("grpc.keepalive_time_ms",360000)
12          });
13  }
```

A csatorna létrehozása során beolvassuk az app.config fájlban található csatlakozási adatokat. Ezeket az adatokat egy socketté állítjuk össze. Ezután beolvassuk a hitelesítést, amit a gRPC névtérben található SslCredentials objektum létrehozásához használunk. Ha ez megtörtént létrehozzuk magát a csatornát. A csatorna bemeneti paraméterei a socket, hitelesítés, és az általunk konfigurált csatorna beállítások. Ezeken a beállításoknak meg kell egyeznie a szerveren lévővel, különben a szerver bonthatja a kapcsolatot hibás konfigurációra hivatkozva. Ezek az opciók arra szolgálnak, hogy a kapcsolat ne szűnjön meg, hogyha a program egésznap is fut, és nem történik egy gRPC hívás sem.

Az fejlécbe történő adat beágyazás nem bonyolult. Ugyanis a gRPC-nél minden híváshoz megadhatók hívási beállítások. Ezek között a találjuk meg a http2 protokoll fejlécét is.

```
1   private CallOptions GetCallOption() {
2           return new CallOptions().WithHeaders(new Metadata()
3                   {
4                           new Metadata.Entry("Authorization", userHelper.
                                Token.Token)
5                   });
6   }
```

Ezért úgy oldottam meg, hogy minden egyes híváshoz lekérünk új CallOptions –t amibe beleágyazom a tokent, amit a bejelentkezéskor kapott meg a felhasználó.

4.2. Szerver implementálása

A szervernek is egy kisebb WPF applikációt hoztam létre, itt nem használtam már semmilyen keretrendszert. A szerverben implementáltam a gRPC protobuffer által generált szolgáltatásokat, MySQL adatbázisban található adatok manipulálásához a szükséges osztályokat, illetve a rendszergazdai kezelő felületet. A kezelő felületek között megtalálható a naplózás beállításainak lehetősége, adatbázis mentése és a felhasználók létrehozása illetve módosítása jelszóval együtt.

Adatbázis kezelő osztályok:

Minden adatbázis kezelő osztályomnak két kötelező interfészt kell implementálnia. Az egyik IDatabaseConnectionManager<T>, a másik pedig a 11 IManage(adat) inte-

részek közül az egyik, ami ugye attól függ, hogy melyik adatnak a manipulációjával fog foglalkozni.

A csatlakozásért felelős interfészt, két metódust, egy függvényt és egy propertyt tartalmazz. Ez egy generikus interfész, így bármelyik adatbázis kezelőnek a csatlakozást megadhatjuk neki (Pl.: MySqlConnection).

```
1 public interface IDatabaseConnectionManager<T>
2 {
3         ConnectionManager ConnectionManager { get;  }
4         void OpenConnection(T connection);
5         void CloseConnection(T connection);
6         T GetConnection();
7 }
```

A property nem más, mint egy ConnectionManager osztály, ami annyit csinál, hogy kiszedi az applikációnk konfigurációs fájljából a szükséges adatbázis információkat. Az OpenConnection és a CloseConnection fog felelni az adatbázis kapcsolat helyes nyitásáért és bezárásért. A bemeneti paraméterek a T osztály. Visszatérési értéke nincs, mivel ha hiba történik valamelyik metódus közben akkor kivételt váltunk ki amit elkapunk és beleírjuk a naplófájlba. A GetConnection pedig az visszaadja a kapcsolatot az adatbázis műveletet végző osztályaink számára.

A másik interfészre egy példa a IManagePartner. Ő sem tartalmazz mást csak azokat a függvényeket amik szükséges a Partner adatainak a manipulációjához. Interfészek használatával meg felek a GOF1-ben foglaltaknak azaz később a Service osztályomba ezekre fogok tudni hivatkozni.

```
1 interface IManagePartner
2 {
3         Partner AddPartner(NewTorzsData newObject, User user);
4         Answer DeletePartner(int id, User user);
5         List<Partner> GetPartnerek(Telephely filter);
6         Answer ModifyPartner(Partner modifiedObject);
7 }
```

Ebben az interfészben megtalálható az AddPartner függvény aminek a bemeneti paramétere egy protobuffer fájlban meghatározott message ami a NewTorzsData ez az osztály tartalmazza a törzs adatokat így bármelyik belehelyezhető. Illetve bekér még egy felhasználót is, erre azért van szükség, mert az adatbázisba csak így tudjuk eltárolni, hogy éppen ki mit csinált. Visszatérési érteke egy Partner típusú objektum. A DeletePartner bemeneti érteke egy partnernek az id-je illetve ugyanúgy bekérem a usert is hasonló szándékok miatt. Visszatérési értéke egy Answer message ami tartalmaz egy string, és egy bool propertyt. Ezeket arra használom, hogy a kliens felé tudom jelezni, ha valami hiba történt és egyből a szervertől fogja meg kapni a hiba pontos leírását is. Letudom kérni a telephelyenkénti partnereket is a GetPartnerek segítségével, így a

bemeneti paraméter egy Telephely osztály lesz a kimenete pedig partnerek listája. Az utolsó függvény a ModifyPartner aminek a bemeneti értéke a módosított partner ami mindent tartalmaz ami számunkra fontos az adatbázisban való módosításhoz. Visszatérési értéke ugyanúgy egy Answer mint a törlésnél.

Az ismétlés elkerülése végett az IDatabaseConnectionManagert egy absztrakt osztályba implementáltam, de már megadtam a generikus változójának az értéket. Ennek az osztálynak a neve MysqlDatabaseManager. Ez egy kis osztály, de annál fontosabb, így minden MySql-el foglalkozó osztálynak csatlakozásáért itt tudom befolyásolni. Illetve ha pontosabb hiba kezelést akarok az adatbázis kapcsolat zárása vagy nyitásához, akkor csak itt fog kelleni módosítani.

Minden IManage(adat) interfészhez létrehoztam egy service osztályt is. Ezek a service osztályok fognak meg jelnni a gRPC szolgáltatásai között, így ha le akarom cserélni az adatbázis osztályt vagy egy másik adatbázis kezelőt kell implementálnom nagyban megfogja könnyíteni a dolgomat, mert ez az osztály dependency injectiont és dependency delegationt használ.

```
1  class PartnerService:IManagePartner
2  {
3          IManagePartner dbManager;
4          public PartnerService(IManagePartner dbManager) {
5                  this.dbManager = dbManager;
6          }
7          public Partner AddPartner(NewTorzsData newObject, User user){
8                  return dbManager.AddPartner(newObject, user);
9          }
10         public Answer DeletePartner(int id, User user){
11         return dbManager.DeletePartner(id, user);
12         }
13         public List<Partner> GetPartnerek(Telephely filter){
14                 return dbManager.GetPartnerek(filter);
15         }
16         public Answer ModifyPartner(Partner modifiedObject){
17                 return dbManager.ModifyPartner(modifiedObject);
18         }
19 }
```

Az osztálynak van egy IManagPartner mezője ezt a mezőt a konstruktorába paraméterként várom. Ebben az osztályban minden metódus a dbManager-re támaszkodik. Ez igazából csak egy közvetítő osztály. Viszont ennek hatására a programom könnyebben tesztelhetővé vált. Ezeknek az osztályoknak már egy teljesen implementált adatbázis kezelő osztályt fogok átadni.

PartnerDatabaseManager ebben az osztályban már az összes metódus az adatbázis műveletekkel foglalkozik, mivel örökölte a MysqlDatabaseManagertől a kapcsolódási

metódusokat. Ezeket fogom átadni inicializáláskor a servicek számára.

Adatbázis kezelő osztályok:

Ez az osztály implementálja az összes olyan szolgáltatást amit a gRPC szervernek tudnia kell. A szolgáltatás osztályunknak az ősosztálya az IktatoService.IktatoServiceBase. Ebben az ősosztályban minden metódus virtuális így az általunk készített osztályban felül írhatjuk őket a saját logikánkkal. A ServiceForgRPC osztály összesen 48 eljárást tartalmazz. Ebből 46 gRPC hívásokkal foglalkozik, míg a maradék kettő közül az egyik a felöltött fájlok beolvasásáért felel, a másik ellenőrzi, hogy a hívásokból érkező fejlécekben érvényes-e a felhasználói adat. A gRPC hívások nagyon egyszerűek nekem így már csak annyi volt a feladatom, hogy meghívtam a MySQLDatabaseManager által szolgálatot valamelyik függvényét és annak a visszatérési értékét csak vissza kell küldenem válaszként. Minden gRPC hívásból bemutatok egyet, kivéve a kétirányú távoli eljárást.

– **ServerStreaming:**

```
1  public override async Task ListIktatas(SearchIkonyvData request,
       IServerStreamWriter<Ikonyv> responseStream, ServerCallContext context)
       {
2          User user;
3          if (CheckUserIsValid(context.RequestHeaders, out user)){
4                  IkonyvService service = new IkonyvService(new
                       IkonyvDatabaseManager());
5                  request.User = user;
6                  List<Ikonyv> ikonyvek = service.GetIkonyvek(request);
7                  foreach (var response in ikonyvek){
8                          await responseStream.WriteAsync(response);
9                  }
10         }
11         else throw new RpcException(new Status(StatusCode.
               PermissionDenied, "Hibas felhasznalo vagy lejart idokorlat!
               Kerem jelentkezzen be ujra!"));
12 }
```

Ennek a függvények a segítségével egy listát fogunk összeállítani a kliens oldalon ezért a paraméterei a következők. Első egy olyan osztály, ami a szűrési paramétereket fogja tartalmazni. Amik a kliens által kijelölt évet, irányt, és a felhasználó id-jét jelenti. A második egy olyan osztály, amivel a kliens oldalra fogjuk tudni küldeni az adatokat. A harmadik a hívásnak a kontextusa. Ebben a kontextusban található meg a fejléc ahol mi a tokent tároljuk, amit a felhasználó kapott a bejelentkezéskor. Megnézzük, hogy a token helyes-e, ha igen akkor kinyerjük belőle a felhasználót, ha nem akkor kivételt váltunk ki ez a kliens felé fog eljutni. Helyes felhasználó esetén tovább megyünk, példányosítunk egy adatbázis kezelő osztályunkból segítségével lekérjük az adatokat ezután

a responseStream.WriteAsync metódussal aszinkron módon visszaküldjük egyesével az adatokat a kliens felé. Ha az utolsó adatot is el küldtük a hívás befejeződik.

– **Unary:**

```
1  public override Task<Answer> ModifyPartner(Partner request,
       ServerCallContext context){
2          User user;
3          if (CheckUserIsValid(context.RequestHeaders, out user)){
4                  PartnerService service = new PartnerService(new
                       PartnerDatabaseManager());
5                  return Task.FromResult(service.ModifyPartner(request));
6          }
7          else throw new RpcException(new Status(StatusCode.
               PermissionDenied, "Hibas felhasznalo vagy lejart idokorlat!
               Kerem jelentkezzen be ujra!"));
8  }
```

A függvényünkkel módosítunk egy partnert. A bemenő paraméter a módosított partner és a hívási kontextus. A visszatérési értéke egy Answer osztály egy példánya, ami jelzi azt a kliens számára, hogy a hívás sikeres volt-e. Példányosítjuk az adatbázis kezelő osztályt és meghívjuk annak az Update függvényét. A függvénynek a bemeneti paramétere egy partner osztály. A visszatérési érteke pedig egy Answer. A Task.FromResult() jelzi a gRPC eljárásunknak, hogy a művelet sikeresen lezajlott és a bemeneti paraméterében megadott értékkel fog visszatérni a klienshez.

– **ClientSreaming:**

```
1  public override async Task<DocumentInfo> UploadDocument(
       IAsyncStreamReader<Document> requestStream, ServerCallContext context)
         {
2          User user;
3       if (CheckUserIsValid(context.RequestHeaders, out user)){
4                  List<byte[]> Chunkes = new List<byte[]>();
5                  Document recivedDocuemnt = new Document();
6                  while (await requestStream.MoveNext()){
7                          recivedDocuemnt = requestStream.Current;
8                          Chunkes.Add(requestStream.Current.Doc.ToArray());
9                  }
10                 DocumentService service = new DocumentService(new
                       DocumentDatabaseManager());
11                 recivedDocuemnt.Doc = ByteString.CopyFrom(Chunkes.ToArray
                       ().SelectMany(inner => inner).ToArray());
12                 Document document = service.AddDocument(recivedDocuemnt,
                       user);
13                 return new DocumentInfo(){
14                         Id = document.Id,
```

```
15                          Name = document.Name,
16                          Path = document.Path,
17                          Size = ((document.Doc.Length / (double)1024) /
                                1024),
18                          Type = document.Type
19                  };
20          }
21          else{
22                  throw new RpcException(new Status(StatusCode.
                        PermissionDenied, "Hibas felhasznalo vagy lejart
                        idokorlat! Kerem jelentkezzen be ujra!"));
23          }
24  }
```

Ez az eljárás a fájl feltöltéséért felel a szerverre. Bemeneti paramétere IAsyncStreamReader és ServerCallContext. Visszatérési értéke egy DocumentInfo osztály, ami tartalmazza a dokumentumnak az összes adatát kivéve magát a dokumentumot. Az olvasó felel az adatok küldésért a szerver felé, ez ugyan olyan, mint amikor a szerver küld a kliens felé egy listát. Megvárjuk, hogy az összes darabkát át küldje a kliens, ha ez megtörtént, akkor az adatot átalakítjuk a gRPC által támogatott ByteStringgé, eltároljuk a szerveren, majd az adatbázisba elmentjük a fájlnak a névét, kiterjesztését, elérési útvonalát és méretét. Ezután összeállítjuk a visszatérési értékünket és kész is vagyunk.

Indítás:

A szerver indításához van lehetőség paraméterek megadására, hasonlóan mint a kliens oldalon. Itt két paraméter elfogadott a –s és a –d. A –d kapcsoló a naplózási szintet debug módba állítja, a –s pedig automatikus el indítja a gRPC szerver szolgáltatását. A fő képernyőm a MainWindow. Innen lehet minden más menüpontot elérni, illetve a szervert ki és be kapcsolni. A szolgáltatás elindításához először be kell tölteni az SSL kulcs párokat, amiket az openssl segítségével generáltam.

```
1   private void StartServer(){
2           try{
3                   SslServerCredentials credentials = CreateCredentials();
4                   server = new Server(new[] {
5                           new ChannelOption("grpc.
                                keepalive_permit_without_calls", 1),
6                           new ChannelOption("grpc.http2.
                                max_pings_without_data",0),
7                           new ChannelOption("grpc.keepalive_timeout_ms",50)
                                ,
8                           new ChannelOption("grpc.keepalive_time_ms"
                                ,360000)
9                   })
```

```
10                  {
11                          Services = { IktatoService.BindService(new
                                Service.SerivceForgRPC()) },
12                          Ports = { new ServerPort("0.0.0.0", Port,
                                credentials) }
13                  };
14                  server.Start();
15                  StartServerButton.IsEnabled = false;
16                  StopServerButton.IsEnabled = true;
17          }
18          catch (Exception ex){
19                  Log.Error("Kovetkezo hiba tortent a szerver indulasakor:
                        {Message}", ex);
20          }
21          Log.Warning("A szerver elindult ");
22  }
```

Hasonlóan a klienshez itt is el kell végeznünk a csatorna beállításokat. A szerver osztály példányosításakor van lehetőségünk megadni a szolgáltatásokat és a portot amin várni fogja a kapcsolatokat. A Services-be megadom a DLL állományban található szolgáltatás egy gyermek osztályát. A portba az ip címet, portot és a kulcspárt adjuk meg. Hitelesítés nélkül is lehet használni a gRPC szervert, de akkor is a portnak át kell adnunk egy osztályt, ami a SslServerCredentials.Insecure. Ez visszatér egy olyan hitelesítési példánnyal, ami nem nyújt, semmilyen biztonsági mechanizmust ezáltal az adataink se lesznek titkosítva a kommunikációs csatornán. Ezután csak megkell hívnunk a szerver.Start() metódusát. Sikertelen indítás esetén például foglalt már a socket a munkaállomáson akkor IOException fog dobni, ha nem keletkezik kivétel akkora szerverünk már képes is fogadni a kapcsolatokat.

Felhasználó kezelés: Ezen a felületen van lehetősége a rendszergazdának felhasználókat hozzáadni, módosítani, illetve jelszavakat módosítani. A módszerek megegyeznek a kliensben látottakkal csak itt, ha beírunk egy új jelszót módosításkor, azaz nem hagyjuk üresen, akkor a jelszó módosításra kerül.

```
1  user = userService.AddUser(new NewTorzsData() { User = user }, new User()
       { Id = 1 });
```

Mivel a szerverben nincs bejelentkezési lehetőség, ezért minden művelet az egy beleégetett felhasználóval végződik, ami az adatbázis scriptbe is belevan égetve. Így tudom, hogy az 1-es id-jű felhasználó mindig az admin lesz. Ebben a kis kód töredékben is ez látható.

Adatbázis mentés:

Itt van lehetőségünk az adatbázisnak a teljes mentésére illetve visszatöltésére. Ha rá megyünk a mentés gombra, felugrik egy útvonal kiválasztó ablak, ezek után meg is kezdődik az adatbázis mentése. Ezt a folyamatot jelezzük a rendszergazdának is

egy indikátor segítségével. A MySql.Data nuGet pacakage tartalmazz egy MySqlBackup nevű osztályt. Ennek segítségével különböző beállítási lehetőségeket lehet megadni az adatbázis mentéshez. Például: a mentet script tartalmazza-e a tábla készítéseket, legyen-e drop utasítás, exportálja-e a nézeteket, függvényeket, tárolt eljárásokat stb.. Az adatbázis visszatöltésénél is hasonló a helyzet.

Naplózási beállítások:

Az utolsó menürészen találhatóak, azok a funkciók, amik a rendszer logolását befolyásolják. Belehet állítani a helyét, szintjét és még azt is, hogy a főoldalon milyen szintű naplózási üzeneteket jelenítse meg. Ezek a beállítási lehetősége jól el vannak különítve a nézeten. Mikor megnyitjuk, ezt a részt a konstruktorban először minden szükséges adatot kiolvasunk a registryből és ezeket az adatokat megjelenítjük a rendszergazda számára. Ha a rendszergazda szeretné módosítani a szerveren mutatott naplózást akkor a következő metódus fog meghívódni:

```
1  private void ModifyCurrentLogLevelToShow_Click(object sender,
       RoutedEventArgs e){
2          if (ServerLoggingLevelToShowComboBox.SelectedItem == null) return
               ;
3          int currentLogLevel = (int)MainWindow.GetLogLevel();
4          int setLogLevelToShow = (int)ServerLoggingLevelToShowComboBox.
               SelectedItem;
5          if (currentLogLevel <= setLogLevelToShow){
6                  RegistryHelper.SetLogLevelToShow(setLogLevelToShow);
7                  MessageBox.Show("A mutatott logolasi szint modosult");
8                  InformationText.Visibility = Visibility.Visible;
9          }
10         else {
11                 MessageBox.Show("A mutatott logolas szintje nem lehet
                       kisebb mint a szerver logolasi szintje.");
12         }
13 }
```

Először is megnézzük, hogy ki jelölt-e valamilyen szintet, ha nem akkor visszatérünk. Ha igen akkor lekérjük, hogy mi a szerver naplózási szintje, mert ha alacsonyabb a szint, mint amit mutatni akarunk annak nincs értelme és nem is lehetséges, mivel a Serilog nem foglalkozik vele. Ha helyesnek találtuk a szinteket, akkor a registryben módosítjuk az értéket.

4.3. Adatbázis

4.3.1. Adatbázis-kezelő

A vágyálom rendszer által megfogalmazottak alapján egyértelmű volt számomra, hogy egy relációs adatbázis kell létrehozni a program használatához. Én a MySQL adatbázis kezelő rendszert választottam. Ez egy nyílt forráskódú rendszer ezért nagy a támogatottsága, szinte minden rendszeren elfut, legyen az linux vagy windows. A fejlett grafikus fejlesztői felületével könnyen és gyorsan lehet készíteni, menedzselni adatbázisokat. Hátránya, hogy gyenge teljesítményű, tehát nagyobb adatbázis forgalom mellett nem ajánlott.

4.3.2. Táblák

Az adatbázisnak a megtervezésekor mindent információt figyelembe vettem, amit az igényfelmérés során összegyűjtöttem.

Évek: Az aktív évet reprezentálja, hogy melyik évre iktatunk. Év zárása után már nem lehet arra az évre iktatni.

User: A programot használó dolgozók adatait tartalmazza. Három oszlopból áll. Felhasználónév aminek a maximális mérete 45 karakterhosszú. Jelszó ami SHA1 kódolással lesz eltárolva, illetve a felhasználó teljese neve. Admin felhasználó automatikusan belekerül az adatbázis scriptből.

Privilege: A felhasználó jogosultsági szintje a programban. Lehet Admin és User. Az admin jogosultsággal rendelkezők a törzseket szabadon szerkeszthetik, felhasználókat adhatnak, módosíthatnak vagy törölhetnek a rendszerben illetve "törölhetnek" iktatásokat. Míg a sima felhasználó iktatáson kívül még partnert, partner ügyintézőt és ügyintézőket tud csak hozzáadni a rendszerhez.

Telephely: Ez jelöli, hogy az adat melyik telephelyhez tartozik az adatbázisban. Ez vonatkozik az iktatásra és a törzsadatokra egyaránt.

Felh_telephely: A tábla tartalmazz felhasználókat és ahhoz kapcsolódó telephelyeket. Ez egy több a többhöz kapcsolati tábla. Így lehet elérni, hogy egyes felhasználókhoz több telephely is tartozzon.

Partner: A telephelyekhez tartozó partnereket tartalmazza. Ezek az adatok az iktatásban fognak nagyobb szerepet játszani.

Partnerügyintéző: A telephelyeken eddig felvitt partnerhez kapcsolódó ügyintézőket tartalmazza. Tartalmazz partner id-t, annak érdekében, hogy összetudjuk kapcsolni melyik ügyintéző melyik partnerhez kapcsolódik.

Partnerügyintéző kapcsoló: Mivel egyes iktatásoknál előfordulhat, nincs partnerügyintézője csak partnere. Így ebbe a táblába tárolódnak el az iktatás partnere és partnerügyintézői. A partnerügyintéző lehet null értékű a táblában.

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 szervezeten belüli dolgozóra, kollégára utal, hogy ezt az ügyet vagy iratot ki intézi.

Doc: Itt tároljuk az iktatásokhoz feltöltött állományok információit. Ami a neve, kiterjesztése, elérési útvonala és egyéb naplózási információk.

Ikonyv_docs: Ez is egy több a többhöz kapcsolat. Mivel egy iktatáshoz akár több dokumentum is kapcsolódhat.

Ikonyv: Ez maga az iktatásokat tartalmazza. Ha bejön egy irat vagy kimegy azt itt lesz rögzítve.

4.3.3. Tárolt eljárások

Minden kérést, amit lehetett tárolt eljárásba tettem. Ezzel is megakadályozva az SQL injectionnek a lehetőségét. Összesen 52 darab lett belőlük. Pár fontosabb eljárást fogok csak bemutatni, mivel a nagy része csak az adatok megfelelő módosításáról, törléséről és hozzáadásáról szól. A legfontosabbak a következők. AddRootIkonyv, AddSubIkonyv, DelIkonyv, setDeletedByValaszID, getNextIktatottID, getIkonyvek. Ezek okozták a legtöbb fejtörést a számomra.

AddRootIkonyv: Bemenete az iktatóhoz szükséges adatok mint a Tárgy, Hivatkozási szám, Ügyintéző id-je, partner, partnerügyintéző id, felhasználó id, telephely, csoport, jelleg,irany id-k és persze a határidő és a érkezett dátum, megjegyzés szövege. Kimenete az új id és az új iktatószám lesz, mivel a többi dolgot a kliens oldalon összetudjuk rakni. Első lépésben lekérem az aktuális évet, erre azért van szükség, hogy az iktatószámhoz hozzátudjam adni. Második lépésben lekérem a következő Id-t. Mivel több telephely adatai is szerepelnek, a táblában 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-jét a táblából így az generateIktSzam viewval le tudom generáltati az iktatószámát és hozzáadni az iktatókönyvhöz.

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

DelIkonyv: 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 DelIkonyvnek. 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 ikonyv where telephely = telephely_b and valaszid is null. Viszont ha van válaszid akkor ugyan ezt a selectet hívom meg csak a where feltétel módosul.

getIkonyvek: Bemente 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 prevYearIkonyv-re is, de ahhoz még kelleni fog bemenő paraméternek az év.

4.3.4. Nézetek

Összesen három darab nézetem van. Ebből egydarab az iktatószámok generálásáért és két darab az iktatások helyes letöltetésért felel. Első a creatIktSzam, ahol az iktatókönyv tábla összes idegen kulcsa össze van kapcsolva a táblájával és ezekből összeállítja az aktuális iktató könyv iktatószámát. A második a currentYearIkonyv itt lekérem az aktuális év iktatásait olyan formában, amiben majd a programban kelleni fog. Illetve az utolsó a prevYearIkonyv ami hasonló a currentYearIkonyvhöz de itt saját kezűleg kell paraméterezni az évet.

5. fejezet

Továbbfejlesztési lehetőségek

A programot lassan egy hónapja használják. Ez alapján kapott visszajelzések azt mutatják, hogy követelmény specifikációban leírtak teljesen lefedték a mindennapi használat során felmerülő funkciókat. Kisebb tovább fejlesztésekre tartanak igényt, mint például az új iktatáskor a tárgy illetve hivatkozási szám mezőben legyen automatikus kitöltési lehetőség, azaz előzőleg felvitt iktatások adataiból lehessen választani. Ez nagyban lecsökkentené a felvitelre szánt időt a program felhasználói számára.

Sajnálatos módon már nem volt időm a következő feladatokra, amiket még terveztem a program készítése során, de a jövőben fejlesztésre kerülnek:

- *Mentés ütemezés:* A szerver oldalra terveztem egy olyan funkciót ahol be lehet állítani, hogy az adatbázisról milyen időközökként készítsen mentést és azt hova helyezze a merevlemezen. Ez által is csökkentve a szerver adminisztrátor terhét. Ebből a funkcióból egyelőre csak az adatbázis mentése funkció került megvalósításra.

- *Cache/Cache kezelés:* Az adatbázis szerver felé irányuló folyamatokat szeretném ezáltal csökkenteni. Egy proxy osztályt fogok létrehozni az adatbázis kezelő osztályokból így azok el fogják tárolni a lekért adatokat, és ha nem volt módosítás a következő kéréskor, akkor ők azt fogják visszaadni. Ezeknek az osztályoknak a kezelését is elérhetővé fogom tenni a szerver oldalon, azaz ha túl nagy lenne egy cache, akkor onnan könnyen lehet majd azokat törölni.

- *Csoportosítás:* A kliens oldalon lévő keresés menüpontnál lévő datagrid-ben az iktatások csoportosítása előzményezés szerint. Azaz ha egy iktatószámnak vannak „gyermekei” akkor az a „szülő” melletti + gomb által lesznek megjeleníthetőek, lenyithatóak.

- *Hiba kezelés bővítése:* A kliensben pontosabb hibaüzenetek létrehozása adatfelvitelkor. Így a felhasználó sem fog kérdésekbe bocsátkozni még akkor is, ha a hiba mivolta teljesen egyértelmű.

6. fejezet

Összefoglalás

A szakdolgozatom kitűzött célja az volt, hogy az Ótemplomi Szeretetszolgálat jelenlegi ügyviteli rendszerét lecseréljem. A jelenlegi helyzetben lévő problémák az iktatással kapcsolatban, hogy a bevezetésben említett egyesülés után az Excel dokumentumok száma megnövekedett. Az iktatáshoz használt munkafüzetek hátránya, hogy az iktatások konzisztenciájának megőrzése lehetetlen, azaz egy-egy be-és kimenő irat is kaphat olyan számot, ami már foglalt, illetve előfordulnak, olyan esetek, amelyek több munkacsoportot is érintenek, emiatt egy iktatást többször vagy egyszer sem iktatnak le. Az iktatásokhoz nem lehet előzményezést rendelni. Az iktatott dokumentumokat csak több perces keresés árán tudják megtalálni. A gazdasági vezető nem képes nyomon követni telephelyek iktatásait, illetve bármilyen dokumentumot elérni. Valamint probléma, hogy a dokumentumokról nem készül semmilyen mentés, archiválás.

Ezekre a hibákra az iktatóprogram fejlesztése hozott megoldást. Most már központosítottá vált. A vezető által kért nyomon követési lehetőség elérhető lett. Mindenki az iktatást egy kliens segítségével éri el, bejelentkezés után. Ezáltal naplózhatóvá vált, hogy ki, mit csinál a rendszerben. Az iktatószám generálását most már a szerver látja el, ezáltal kiküszöböltük a számok többszöri kiadását. Egy átláthatóbb felületet kaptunk az iktatás felvitelére, keresésére és a törzsadatok kezelésére. A felvitelkor a már tárolt törzsadatok biztosítják a pontos, precíz adatok kitöltését, illetve egyes felhasználók több telehelyhez is rögzíthetnek iktatást. A kereséskor csak a legfontosabb adatokat jelenítjük meg. Elérhetővé vált a paraméteres keresés és szűrés. Az iktatásokhoz megvalósult a kapcsolódó dokumentumok egy helyen való kezelése is, így most már egyszerűbb ezen adatok megtekintése, feltöltése és letöltése.

A szerver oldal elérhetővé tette számunkra az adatbázis mentését, és annak archiválását. A felhasználók egyhelyen való kezelését. Ahol az adminisztrátor hozzárendelhet, illetve eltávolíthat telephelyeket a userektől. Ezt a lehetőséget admin jogosultsággal rendelkező felhasználó is eléri a kliens oldalról, viszont jelszó módosításra csak itt van lehetőség. A logolás helyét és szintjét is beállíthatjuk. A szint alapértelmezetten warningon van, azaz csak olyan hibák fognak mentésre kerülni, amik nagyban befolyásolják a

program működését. Ha a klienst paraméteresen (-d) indítjuk akkor az automatikusan debug módra vált. A szerver és a kliens között ez TLS kapcsolat van SSL titkosítással.

Az eddigi tapasztaltak alapján a kollégák nagy örömmel fogadták a programot. Kaptam tovább fejlesztésre való ötleteket illetve kisebb javítási igényeket a felhasználó felületben. Úgy érzem, a gazdasági vezető által kitűzött, és a felhasználók munka végezésének megkönnyítésére tett célokat elértük.

Irodalomjegyzék

[1] Fazekas István: *Valószínűségszámítás*, Debreceni Egyetem, Debrecen, 2004.

[2] Tómács Tibor: *A valószínűségszámítás alapjai*, Líceum Kiadó, Eger, 2005.