1. **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 végezz, 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 átadtak egyházaknak. Ennek következményében a Szociális- és Gyermekvédelmi Főigazgatóság fennhatósága alatt álló Békés Megyei Körös-menti Szociális Centrum szarvasi telephelye átadásra került a Magyarországi Evangélikus Egyház szarvasi diakóniájához. Az intézmény így az Ótemplomi Szeretetszolgálat részévé vált. Mivel egy időben több telephellyel is bővült, ezért szükségessé vált az ügyviteli, leltározási és nyilvántartó rendszereinek felülvizsgálata. Az eddig 50 fővel működő diakónia létszáma megengedte az Excel használatát, viszont a megtöbbszöröződött létszámmal a dokumentumok átláthatatlanabbak, kezelhetetlenebbek lettek és 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 volt lehetőségem kurzus formájában felvenni 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-t, 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. 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.

Excel hibái a jelenlegi helyzetre

1. A legtöbb ember nem szereti

Valamilyen oknál fogva az emberek többsége nem szereti az Excelt, főleg azok a dolgozók között, akik nem használják gyakran. Legtöbben zavarónak és félelmetesnek találják. Ez nálunk sincs másképp, hasonló problémákba ütközök a nap, mint nap a kollégáim körében. Ha valaki kap egy feladatot, amit Excelben kell elvégeznie a legapróbb nehézségnél is segítséget kérnek.

1. A fontos adat eltűnik

Mivel az összes adatot látod egyszerre, ezért nagyon nehéz átnézni úgy, hogy felfedezd 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 ezek egy idő után zavarókká tudnak válni.

1. 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.

1. 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 osszák meg 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 elvész a dolgozók beérkezett üzenetei között.

1. 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. Nem lehet dokumentumokat csatolni az egyes iktatásokhoz így annak visszakeresése nehézkes.

1. **Követelmény specifikáció**

Fogalomtár: ( Néhány fogalom netjogtár)

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 az 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: Az iktatás: az 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 fizikai jellemzője. 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 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 adategyüttes

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.

hivatkozásiszám: A beérkezett irat iktatási száma.

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

* 1. 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 egymás iktatási anyagát, így egymás segítségére szorulnak. Az iktatott dokumentumokat csak több perces keresés árán tudják megtalálni. A gazdasági csoport vezetője szeretné látni az összes telephely iktatását, de a jelenlegi helyzetben ez nem megoldott.

* 1. Vágyálom rendszer

Egy olyan központosított iktatás létrehozása, ahol a személyenkénti gondolkodásmódot elhagyhatjuk. Az iktatókönyv összeállítása egyszerű és könnyen kezelhető, valamint a generált iktatószám azonnal olvasható legyen. Ne kelljen ügyelni arra, hogy ki és mikor iktat, a program automatikusan generálja a megadott adatokból az iktatószámot. El lehessen különíteni telephelyeket, illetve legyen olyan lehetőség is, hogy egyes felhasználókhoz több telephely is tartozzon vagy mind. Törzsadatokat, mint partner, ügyintéző, jelleg, csoport, telephely, felhasználók tudjon kezelni a program. Bizonyos beállításokat csak admin végezhessen el, mint felhasználó, telephely hozzáadása/módosítása és iktatások törlése, év nyitás/zárás. Évenként elkülöníthetők és lezárhatóak legyenek az iktatások. Irányuk lehet bejövő és kimenő jellegű. Az iktatószámot úgy generálja le a program, 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 telehelyhez 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. A dokumentumokat gyorsan és egyszerűen fel lehessen tölteni az iktatásokhoz. Ezek jellege Word, Excel, PDF. Könnyen lehessen keresni a felvitt adatok között. Jó lenne egyházi jelleg, például a napi ige megjelenítése a főoldalon. A rendszergazdának legyen szerveren kezelő felülete, ahol az adatbázist, felhasználókat és a logolást könnyen tudja kezelni. Feleljen meg az iratkezelésre vonatkozó jogszabályoknak és az intézményi ügyiratkezelési szabályzatnak. A legfőbb cél, hogy az iratkezelés legyen szakszerű, gyors, gazdaságos és naprakész.

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

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:…

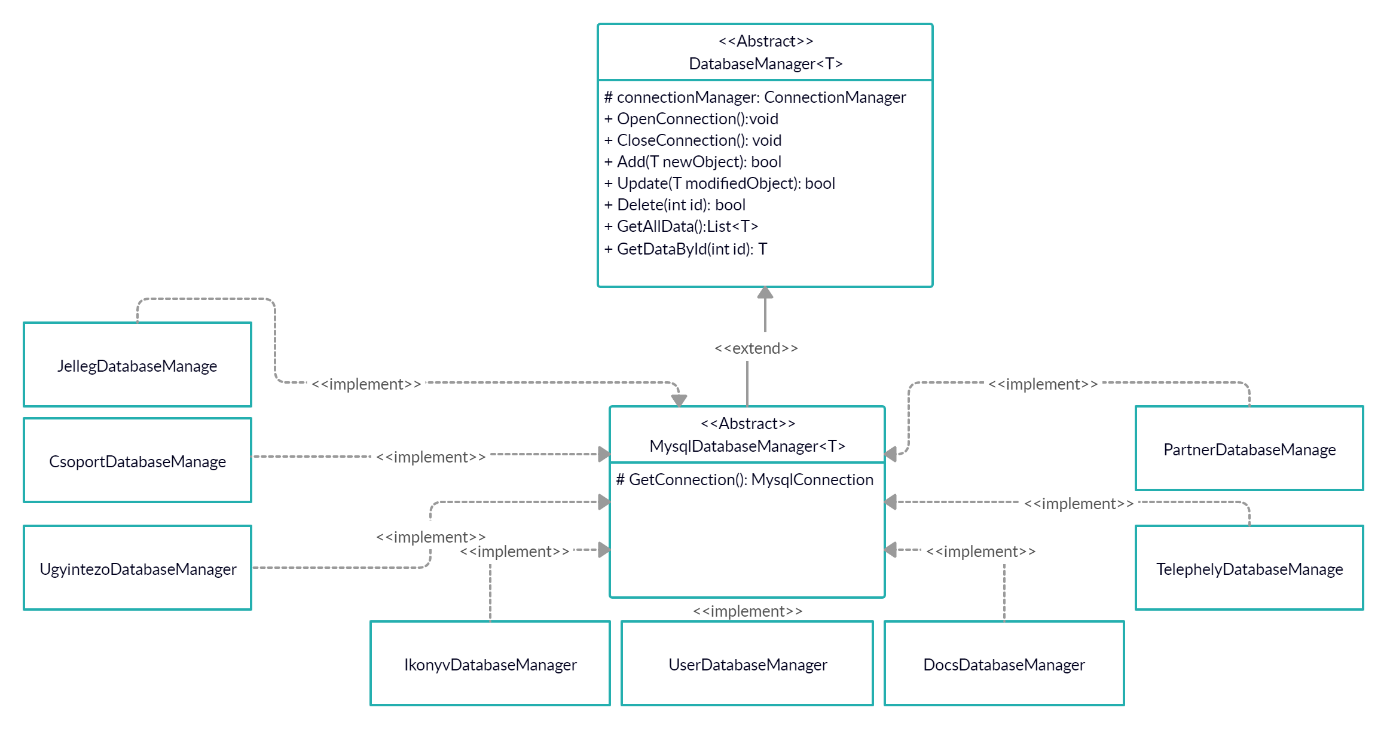
* 1. Követelmény lista

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Név | V. | Kifejtés |
| 1 | Iktatás kezelhetősége | 0.2 | Egyszerű iktatás, az iktatáshoz tartozó információk egy legördülő menüben kiválaszthatóak legyenek (ügyintéző, jelleg, csoport, partner, partnerügyintéző, irány). A tárgynak és a hivatkozási számnak legyen külön beviteli mezője, csak aktuális évre lehessen iktatni. Előzményezés hozzárendelésének lehetősége az iktatáshoz. Az iktatásokat törölni csak admin jogosultsággal lehessen. Az iktatások bárki által módosíthatóak, de csak azokat az információkat, amik nem befolyásolják az iktatószámot. Két dátum legyen, érkezett és határidő és lehessen megjegyzést fűzni az iktatáshoz. Egyedül a hivatkozási szám és a partnerügyintéző ne legyen kötelező mező. |
| 2 | Bejelentkezés | 0.4 | Legyen egy szép és egyszerű bejelentkezési felület, ahol a felhasználó nevet el lehessen menteni, de csak akkor, ha szeretné a user. |
| 3 | Törzsadatok felügyelete | 0.1 | Egy helyen lehessen kezelni a törzset. Mindegyik törzshöz lehessen 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 | Keresés az iktatások között | 0.3 | A keresés szűrhető legyen évre és irányra. De az iránynál az össze is látszódhasson. Keresni lehessen iktatószámra, tárgya, partnerre, jellegre, ügyintézőre, csoportra és hivatkozási számra. A kereséskor is lehessen feltölteni dokumentumot és módosítani az iktatást. |
| 5 | Dokumentum feltöltés | 0.6 | Minden iktatáshoz lehessen feltölteni dokumentumokat. Dokumentumok jellege lehet PDF, Word, Excel. Ezeket egyszerűen vissza lehessen nézni, és ha kell törölhetőek legyenek. |
| 6 | Jelszavak biztonsága | 0.5 | A felhasználók jelszavát a rendszergazda se tudja visszanézni, viszont módosítani csak ő tudja. A jelszavak legyenek titkosítva az adatbázisban. |
| 7 | Logolás beállításai | 0.7 | A szerver oldalon állíthatóak legyenek a logolás szintjei. Egyszerűen lehessen módosítani a log fájl elérési útját. |
| 8 | Adatbázis műveletek | 0.8 | A szerveren legyen lehetőség az adatbázis mentésére és annak visszatöltésére. Az adatbázis mentés ütemezhető legyen. |
| 9 | Felhasználó kezelés | 0.9 | A szerver oldalon is legyen lehetőség a felhasználó felvitelére, módosítására és törlésére. A jelszavak módosítása csak innen legyen lehetséges. |
| 10 | Főoldal kinézete | 1 | A belépés után jelenjen meg az üdvözlő felület. Ahol megtalálható az egyház szimbóluma, és a napi ige. |

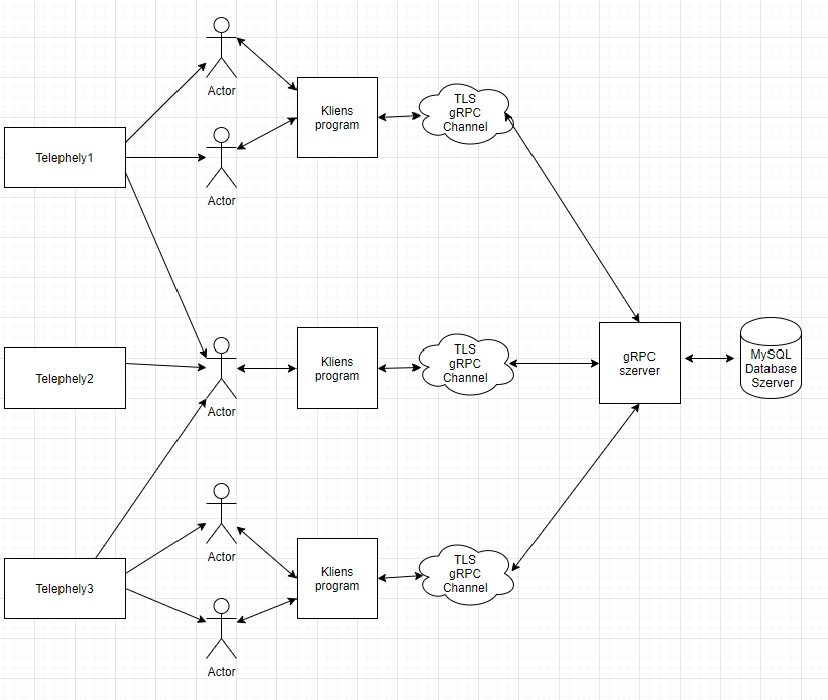
* 1. Technikai feltétel

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

1. Rendszerspecifikáció







A rendszerrel szemben támasztott általános követelmények

* A rendszer funkcióit csak bejelentkezés után használhatják a felhasználók
* Adattárolás MySQL adatbázison
* A szerver és a kliens Windows platformra legyen tervezve

Az alkalmazással szemben támasztott funkcionális követelmény

Felhasználókezelés

* + Admin
  + User

Funkcionális specifikáció

Alkalmazás funkciói

* 1. **Userek számára elérhető funkciók**
     1. **Bejelentkezés:**

A regisztrálás a rendszergazda vagy admin által történik. 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. Kilépés gomb bezárja a programot.

* + 1. **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 200 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.

* + 1. **Iktatás módosítása:**

Minden felhasználó tud az iktatáson módosítani, de csak azokat az adatokat, amik magát az iktatószámot nem befolyásolják.

* + 1. **Dokumentum kezelés:**

Itt megjelennek a hozzáadott állományok, ezeket egy listában mutatjuk meg a felhasználó számára, ahol feltüntetjük a nevüket, méretüket, kiterjesztésüket és itt jelenik meg a törlés gomb. 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.

* + 1. **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.

* + 1. **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

* 1. **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. **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.

* + 1. **Törzsadatok:**

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

**Felhasználó:**

Itt van lehetőség felhasználó felvitelére, módosítására és törlésére.

***Hozzáadás:*** 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 esélyünk a felvitt telephely törlésére. A törölt adat visszakerül az eredeti helyére.

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

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

* + 1. **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.

1. **MVVM**
   1. **MVVM bemutatása:**

Az MVVM nem mást jelent, mint Model, View és ViewModel ami egy architetkurális tervezési minta. Ezt a mintát először a Smalltalk összefüggésében írták le a Xeroxnál, 1979-ben. Ez a minta elválasztja a GUI (Graphica l user interface)-t az üzelti logikától. Kliens oldalon szokták használni. Ebben a mintában, a model nem jelent mást, mint az adat, amit meg fogunk jeleníteni a view-ban a viewmodelnek 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 code-behind-ban.



A ViewModel úgy viselkedik, mint egy közvetítő a view és a model között, és felelős a view logikájának a kezeléséért is. Általában, a viewmodel interakcióba lép a modellel azáltal, hogy meghív egy metódus a model osztályban. Ez után a viewmodel biztosítja az adatot a view számára olyan formában, hogy a view azt könnyedén tudja használni. A viewmodell az adatokat a modellből szedi ki és aztán azokat elérhetővé teszi a view számára, előfordulhat, hogy ezeket az adatokat egyszerűbbé kell formázni, hogy a view kezelni tudja őket. A viewmodel implementálja azokat, parancsokat is amivel a felhasználó interakcióba tud lépni a viewban. Például ha a felhasználó rá klikkel egy gombra a UI az triggerel egy metódust a viewmodelben. A viewmodelnek még olyan feladata van, hogy a logikai állapotokat definiálja.

* + 1. **A Nézet és a Nézetmodell**

A view és a viewmodel adat kötés, metódusok hívások, propertyk, eventek, és üzeneteken keresztül kommunikálnak egymással. A viewmodel nem csak a model adatait tartalmazza hanem más propertiket is mint például állapot információk például töltünk e valamit vagy a gomb megnyomható-e és parancsokat amik olyan funkciók amiket eventek kezelésére alkalmazunk mint a gomb megnyomása. A nézet csak a saját UI eventjeivel foglalkozik. Amiket a viewmodel segítségével térképezz fel. A modellek és a propertyk a viewmodelben a view-on keresztül lesznek módosítva két irányú adatkötés által.

* + 1. **A NézetModell és a Modell**

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

* A viewmodel mutathat a modellre közvetlenül vagy propertyken keresztül.
* A viewmodel tartalmazhat interfészeket szolgáltatásokhoz, konfigurációs adatokhoz, amellyel tudja a propertijeit feltölteni, módosítani, amik a view-ra mutatnak.
  1. **MVVM keretrendszer** 
     1. **Caliburn Micro bemutatása:**

Ez egy kicsi mégis erőteljes keretrendszer, amit arra terveztek, hogy megkönnyítse a programok fejlesztését minden XAML platformon. Erős MV\* pattern támogatottsága segít abban, hogy a megoldásaidat gyorsan, kód minőség vagy tesztelhetőség feláldozása nélkül fejleszthessük. Fejlesztők: Nigel Sampson, Rob Eisenberg és Thomas Ibel. Mivel ez is egy open source projekt ezért sokan mások is hozzájárultak a fejlesztéshez a teljes listát a githubon meglehet tekinteni. A Caliburn.micro is minden sok más keretrendszer is név konvenciót alkalmaz. Ezáltal találja meg a viewmodel a hozzá tartozó view-t. 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:

* + 1. **Action Messages**

Az Action működése megengedi számunkra, hogy összekössük a UI triggereit mint például gomb nyomása a viewmodelben található metódussal. Ez a mechanizmus lehetőséget nyújt paraméterek továbbítására is a metódus számára. A paraméter lehet egy másik FrameworkElement érétke vagy más speciális érték, mint a DataContext vagy EventArgs. Minden paraméteren el végez egy automatikusan típus konverziót, ami a metódus szignatúrájához viszonyul. 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 triggere használhassuk.

* + 1. **Action Conventions**

A konvenciók a x:Name-re épülnek. Szóval ha van egy metódusunk amint „Login”-nak hívunk és van egy gombunk UI-on aminek a neve ugyanúgy „Login” akkor arra a keretrendszer automatikusan létrehoz egy EventTriggert-t a gomb click eventjre és hozzárendeli az ActionMessage a Login metódushoz. 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 más eventkehez is rendelhetunk funkctiókat megnevezés alapján. Például, tudunk olyan eseményre is feliratkozni, mint a MouseMove.

* + 1. **Binding Conventions**

A Caliburn.Micro támogatja a konvenció-alapú adatkötést is. Ez ugyanúgy x:Name segítségével is működik. Hogyha van egy property a viewmodellünkben ugyan olyan névvel, mint az elem neve akkor azt megpróbálja összekapcsolni. Mikor egy bindolás bekövetkezik akkor a framework több lépésen keresztül építi a kapcsolatott. Ezek közül mind testre szabható, mint például a bindolás módja azaz BindingMode vagy StringFormat, ValueConverter stb… .

* + 1. **Screen, Conductor és Collection**

A Screen, Conductor és Collection osztály engedélyezi számunkra a model-alapú követését az aktív vagy a jelenlegi viewnak, kikényszeríti a screen életciklust és a helyes leállítást vagy leállítás megszakítást az applikációban. A screen usercontroll viewmodelleként használják. Ezek az építő elemei a megjelnítésnek a caliburn.microban.A conductor és Collection is a Screen osztály leszármazottjai. A Conductor is lehet használni usercontrolloknál de ezt már windowok vagy pageknél alkalmazzák, lényege, hogy képes kezelni egy Screen típusú osztályt, amit aztán meg tud jeleníteni. A Collectionek két fajtája van mindegyik több screen befogadására képes, de az egyik egyszerre csak egy screent jelenít, meg míg a másik akár egyszerre az összeset is.

* + 1. **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 threaden történik. Támogatja a polimorfizmust is.

* + 1. **View Locator**

Minden ViewModelhez a programunkban van egy alap stratégia a hozzá tartozó view megtalálásához. Ezt név konvenció alapján végzi el. Például ha a viewmodelleünket elnevezzük IktatogRPCClient.ViewModels.ContainerViewMode-nek akkor a Caliburn.Micro a IktatogRPCClient.Views.ContainerView-t fogja keresni. Egy view-nak akár több viewmodelje is lehet.

* + 1. **View Model Locator**

Attól függetlenül, hogy alapértelmezetten a Caliburn.Micro a ViewModel-től indul ki alapértelmezetten ez módosítható. Azaz támogatja, hogy először a view-t keresi és utána viewmodelt ugyan azzal az eljárással mintha ennek az ellenkezője lenne.

* + 1. **Window Manager**

Ez a szolgáltatás a is jelzi, hogy ez a keretrendszer View-Model központú. Átadunk a számára egy példányt abból a ViewModelből amit megszeretnénk jeleníteni, ő megkeresi a hozzátartozó nézetett, beállítja az összes konvenciót amit beállítottunk rajta és megjeleníti az ablakot.

* + 1. **PropertyChangedBase and BindableCollection**

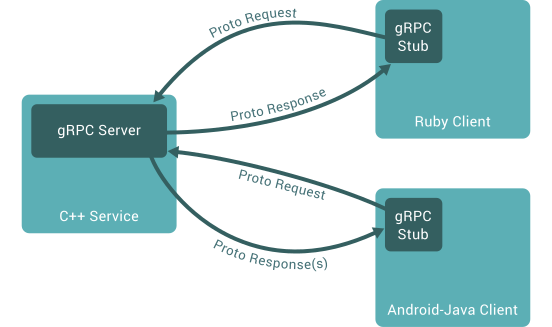
A frameworkben lévő implementáció lehetővé teszi számunkra, a string és lambda kifejezés alapú változás értesítéseket. Megbizonyosodik róla, hogy minden event ami a propertyhez kapcsolódik a UI threaden történjen. A BindableCollection az ObservableCollection leszármazottja, azaz egy egyszerű lista azonban itt is ügyel arra, hogy minden eventje az UI threaden történjen.

* + 1. **Bootstrapper**

Ennek az osztálynak a segítségével tudjuk a szoftverünket elindítani és futtatni. Nem kell, mást tenni hozzá csak létrehozunk egy saját osztályt ami a Bootstrapper osztály leszármazottja és hozzáadjuk az a program ResourceDictonary-be. Innen tudjuk begyűjteni még az indítási paramétereket is.

1. **gRPC és Protobuf**
   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 főrészből 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 azaz ha a szerver oldalt megírom c++-ban akkor az fog tudni kommunikálni bármilyen az általa támogatott nyelvben írt gRPC klienssel. Támogatott nyelvek:c#, java, c/c++, Dart, Go, Kotlin, Node, Objective-C, PHP, Python, Ruby. Egyik előnye, hogy automatikusan generálja nekünk a kliens stubokat és az abstract szerver osztályokat.



* + 1. A 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 stub ami azoknak a metódusoknak a fejlécét tartalmazza amik a szerveren is megtalálhatók. Ezek segítségével fogjuk meghívni a szerver oldalon a service által definiált metódusokat.

* 1. **Protobuffer bemutatása**

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

* + 1. **Message**

A message egy üzenet ami az rpc híváskor adható meg paraméterként illetve ilyen üzeneteket tud visszaküldeni a szerver. Itt lehet a classokat kialakítani illetve egyéb segéd osztályokat. Továbbá az rpc hivásoknál nem lehet üres a metódus ezért létre kell hozni egy EmptyMessaget aminek nincs semmi mezője és ezt kell megadni paraméternek. Megtalálhatóak benne skalár típusok 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.

* + 1. **Service**

A service nem más mint azoknak a RPC metódusoknak gyűjtő osztálya amit használni szeretnénk. Egy gRPC szerver több servicet is ki tud szolgálni egyszerre egy adott socketen. Alapból a compiler generál egy abstract interfacet és egy hozzá tartozó típus biztos stub implementációt. A stub fogja továbbítani a hívásokat egy RPCChannel-en keresztül, amit mi a szerver oldalon testre szabhatunk.

* + 1. **A gRPC RPC tipusa**

**Unary**

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

**Client streaming RPC:**

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 RPC:**

Ez a kliens streamingnek a fordítottja, de itt ugyanúgy a kliens kezdi a kommunikációt küld egy üzenetet erre a szerver több üzenettel válaszol.

**bidirectional streaming rpc**

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

* + 1. **Protobuffból 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 abstract 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 dll állományokat. Ezeket az állományokat már könnyedén hozzáadhatjuk a szerver és a kliens oldali projektünkhöz

1. **Implementálás:**
   1. **Kliens implementálása Caliburn.Micro:**

Ebben a részben befogom mutatni a kliens oldali programom részeit. Mivel elég sok view-t és viewmodelt tartalmaz ezért csak főbb összetevőit és problémásabb részeket fogom kiemelni. A kliens több nézetből áll ezek közül a legfontosabb a bejelentkezés, konténer, főoldal, iktatás, keresés, törzsek kezelése. Ezeknek a nézeteknek fogom bemutatni röviden a viewmodeljeit és azok metódusait és propertijeit. A kliensre a Caliburn.Micro és a gRPC is elérhető a nuGet package manageren keresztül.

**Indítás:**

Az indításhoz való legelső konfigurálni valónk nem más volt, mint az applikáció ResourceDictionary én sem tettem mást, mint a dokumentációkban leírtak alapján hozzáadtam az indító fájlt illetve kivettem a Startup részt az application részből.

<Application.Resources>

<ResourceDictionary>

<ResourceDictionary.MergedDictionaries>

<ResourceDictionary>

<local:Bootstrapper x:Key="Bootstrapper"></local:Bootstrapper>

</ResourceDictionary>

<ResourceDictionary Source="style.xaml"/>

</ResourceDictionary.MergedDictionaries>

</ResourceDictionary>

</Application.Resources>

Mint látható a ResourceDictionaryhoz egy MergedDictionariet adtam hozzá, így a saját szótáram is hozzáadható és nem kell minden formázási stílust a XAML fájlba közvetlenül beleírnom és újra felhasználhatóvá válik. Látható még az is, hogy itt lesz megadva, hogy melyik fájl legyen az alapértelmezetten indított ami a Bootstrapper.

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 public-nak 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 állítjuk be alapállításba. 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 viewmodelt. Ezután a korábbiakban leírt módon meg keresi a hozzá tartozó view-t és megjeleníti azt.

**Bejelentkezés:**

Ebben a viewmodelben összesen három property található. Ezekbe tárolom el a felhasználó adatait illetve, hogy a felhasználó szeretné-e, hogy megjegyezzük a jelszavát a következő induláskor. Ha a szeretnénk eltárolni a felhasználó nevet akkor azt 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} Csatlakozás a szerverhez", GetType());

bool success = await userHelper.Login(GetDataFromLoginTextBoxes());

if(success)Log.Debug("{Class} Sikeres csatlakozás", GetType());

else Log.Debug("{Class} Sikertelen csatlakozás", GetType());

if (SaveUsernameIsChecked) Registry.SetValue(keyName, "Felhasználónév", UsernameBox);

else Registry.SetValue(keyName, "Felhasználónév", "");

return success;

}

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. Hibás jelszó esetén a gRPC szerver kivált egy exception-t a kliens oldalon és azt jelezzük a felhasználónak.

**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 CreatScene metódusa aminek egy paramétere a Scenes enumok közül választható. Visszatérési érteke egy screen osztály béli objektum. A Scenes enumok nem mást tartalmaznak, mint a megjeleníthető viewok listáját.

private void ChangeScene(Scenes scene) {

Log.Debug("{Class} Oldal váltása {Scene}", GetType(),scene);

ActivateItem(SceneManager.CreateScene(scene));

}

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 view-ba található contentcontrol framework elementbe 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 viewmodelje az ige modelt 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 a mai nap dátumát. Az utolsó pedig maga a napi ige tárolására szolgál. A főoldalon ezt a modelt statikusként állítottam be, annak érdekében, hogy a viewmodel ne végezze el az ige adatok letöltését és formázást csak ha szükséges. A letöltéshez a következő metódust használom:

private async void DownloadIge()

{

try

{

string result = "";

using (WebClient client = new WebClient())

{

client.Encoding = Encoding.UTF8;

result = await client.DownloadStringTaskAsync("https://napiige.lutheran.hu/igek.php");

}

\_ige.NapiIge = BuildDailyIge(result);

\_ige.LetoltottIgeDatuma = DateTime.Today.Date;

NotifyOfPropertyChange(()=>NapiIge);

}

catch (WebException e)

{

InformationBox.ShowError(e);

}

}

Az aszinkronitás itt is megvan azért, hogy a UI threadet 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:**

Ebben a viewmodelben nagyon sok mező, property és metódus található ezért csak az alap működését fogom bemutatni. A szükséges adatok letöltését a szerverről rögtön a konstruktorból meghívom.

private async void LoadData()

{

Log.Debug("{Class} Adatok betöltése a szerverről.", GetType());

SelectedIrany = Iranyok.First();

eventAggregator.Subscribe(this);

AvailableTelephelyek = await serverHelper.GetTelephelyekAsync();

if (AvailableTelephelyek.Count > 0) SelectedTelephely = AvailableTelephelyek.First();

Log.Debug("{Class} Sikeres adat letöltés.", GetType());

}

Az irányok hard kódolva vannak a programban mert ezek nem fognak változni. Alapértelmezetten kijelöli a „bejövő” részt tehát erre fog iktatni a felhasználó, ha ezt nem módosítja. Itt iratkozok fel a Caliburn.Micro által nyújtott EventAggregatorra. Az eventAggregator segít abban, hogy ha egy IHandle<Ikonyv> vagy IHandle<RemovedItem> event bekövetkezik akkor ez a class kezelni tudja. Az IHandle interfaceket implementelni kellett, hogy ebben a helyzetben mit vigyen véghez a program. Mivel sok helyen előfordul hasonló esett ezért ezeket az interfaceket kiszerveztem egy IkonyvHandlerModel-be, így mindig tudom, miket kell implementálni. A Handlerek implementációja:

public override void Handle(RemovedItem message)

{

if (message.Item is Ikonyv)

{

\_recentlyAddedIkonyvek.Remove(message.Item as Ikonyv);

NotifyOfPropertyChange(() => RecentlyAddedIkonyvek);

}

}

A RemovedItem handler arra szolgál, ha például a keresésben a felhasználó kitörölné az éppen felöltött iktatást, akkor az a statikus Iktatókönyvek közül is kitörölje. Így ha vissza tér az iktatási felületre, akkor már nem lesz közte.

public override void Handle(Ikonyv message)

{

\_recentlyAddedIkonyvek.Remove(SelectedIkonyv);

\_recentlyAddedIkonyvek.Add(message);

NotifyOfPropertyChange(() => RecentlyAddedIkonyvek);

}

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

public void ModifyIkonyv() {

if (SelectedIkonyv == null) return;

WindowManager windowManager = new WindowManager();

Screen screen = new PopUpViewModel(new ModifyIkonyvViewModel(\_selectedIkonyv));

windowManager.ShowDialog(screen, null, null);

}

Ez a metódus a IkonyvHandlerModel-ben található, mindenki ugyan ezt fogja használni. Itt található a WindowManager ami az új ablak megnyitásáért felelős a Caliburn.Micro keretrendszerben. Nem kell mást tennünk mint példányosítunk egyet és a ShowDialoggal el indítjuk az új ablakot. Így ha szükségünk van a dialog.result-ra akkor azt is megkaphatjuk. Nekem jelenesetben nincs rá szükségem mivel EventAggregatort használom. A windowManagernek van még ezen kívül két metódusa az ablakok megjelenítésére a ShowPopup és a ShowWindow ezek már csak voidok.

**Keresés:**

**Törzs:**

* 1. **Szerver implementálás**

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

1. **Adatbázis** 
   1. **Válaszott adatbázis-kezelő**
   2. **Táblák:**

Az adatbázisnak a megtervezésekor mindent információt figyelembe vettem amit az igényfelmérés során összegyűjtöttem. Védelmi szempontok miatt, minden adatbázis műveletet stored procedure-be tettem, hogy az SQL injection-t elkerüljük.

* + 1. **Évek:**

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

* + 1. **User tábla:**

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.

* + 1. **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.

* + 1. **Telephely:**

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

* + 1. **Felhtelephely:**

Minden felhasználóhoz tartozik egy vagy több telephely ahová tud iktatni vagy törzs adatokat rögzíteni.

* + 1. **Partner:**

Azokat a partnereket tartalmazza akiket a

* + 1. **Partnerügyintéző:**

A megadott partnerhez tartozó ügyintézőket tartalmazza

* + 1. **Partnerügyintéző kapcsoló:**

Az a személy, intézmény vagy cég aki küldte az iratot. Munkaszerződéseknél a partner a dolgozó nevét jelöli. Lehet például E-on, Járási hivatal..stb. Ennek a táblának az id-je fog idegen kulcsként megjelenni az iktatásban.

* + 1. **Csoport:**

Az iratok azon típusait jelöli, amely egységhez kapcsolódik az iktatandó anyag. Például Ellátotti, Főzőkonyha, Munkaügy.

* + 1. **Jelleg:**

A dokumentum formai megjelenésének megadása. Ez lehet e-mail, küldemény, fax, levél, munkaügyi irat.

* + 1. **Ügyintéző:**

Ez a szervezeten belüli dolgozó kollégára utal, hogy ezt az ügyet vagy iratot ki intézi.

* + 1. **Doc:**

Itt tároljuk az iktatásokhoz feltöltött állományokat mediumblobban. Illetve eltároljuk még annak nevét és a kiterjesztését is. Lehet Pdf,JPG,PNG,XLSX,DOCX...stb.

* + 1. **Ikonyv docs:**

Az adott iktatáshoz tartozó dokumentum.

* + 1. **Ikonyv:**

Ez maga az iktató könyv. Ha bejön egy irat vagy kimegy azt itt lesz rögzítve. Az iktatószámot tárolt eljárással fogom előállítani ami a megadott adatok alapján fog generálódni. Egy példa: **B-SZ/R/3/2019 Ennek felépítése**

Az első karakteret az határozza meg, hogy K - kimenő vagy B - bejövő

A második karakter a jellege határozza meg SZ pl szerződés.

A harmadik karakter a telephely jelöli pl. R - Rákóczi, V- Vajda stb..

A negyedik karakter sorozat a sorszám ami lehet kötőjeles Pl. B-SZ/R/3-1/2019 vagy B-SZ/R/3-1-1/2019 a válaszokhoz mérve.

Az utolsó rész pedig az évet jelöli.

A generált iktatószám el is lesz tárolva a táblában.

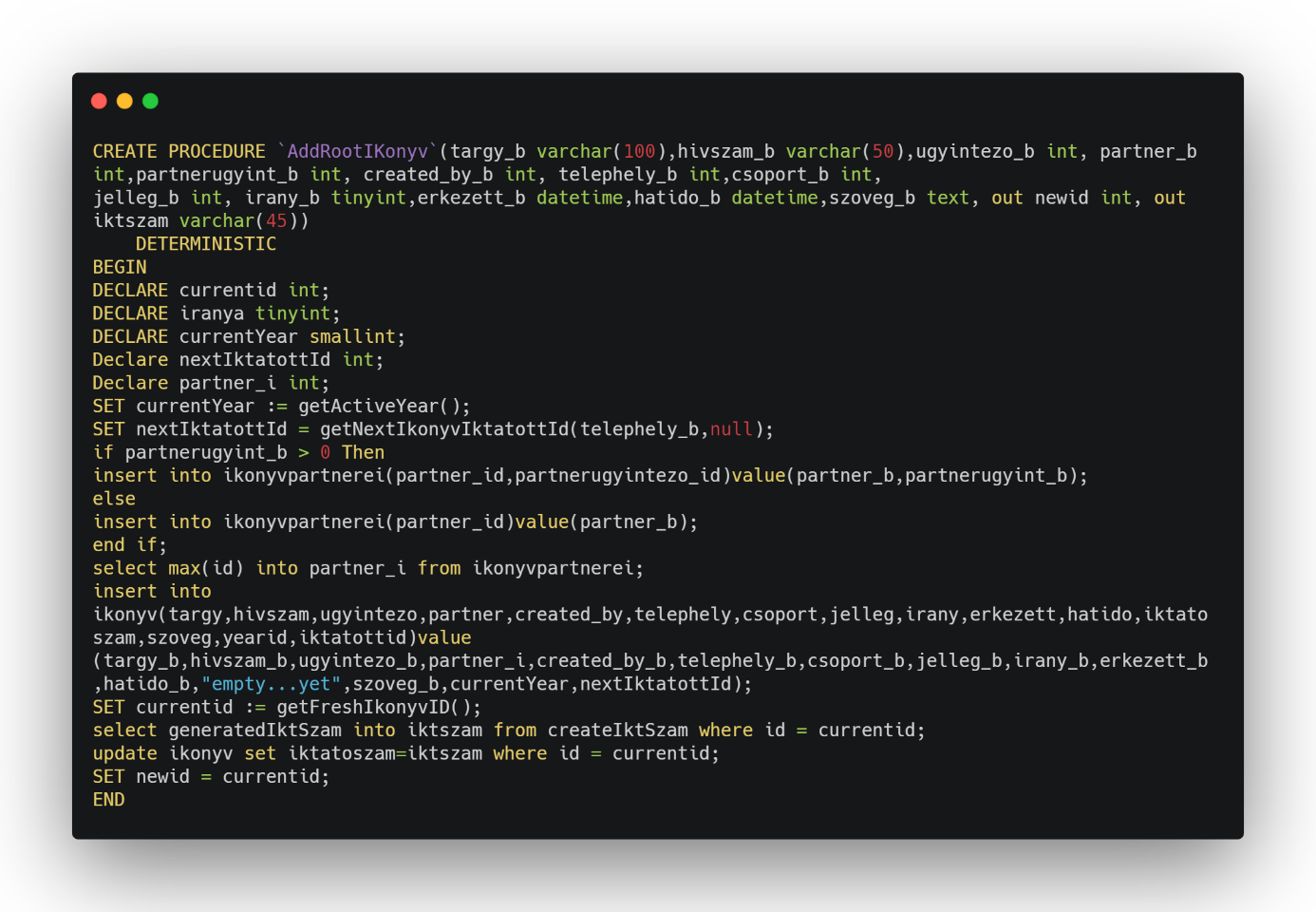
* 1. **Tárolt eljárások**

Minden kérést amit lehetett tárolt eljárásba tettem. Ezzel is megakadályozva az SQL injectionnek a lehetőségét. Összesen 39 darab lett belőlük, de ezeknek a száma valószínűleg csak növekedni fog az iktató program fejlesztése során mivel előfordulhatnak olyan lekérdezések amikre még nem gondoltam.

Pár fontosabb eljárást fogok csak bemutatni, mivel a nagy része csak az adatok megfelelő módosításáról, törléséről és hozzáadásáról szól. A legfontosabbak a következők.

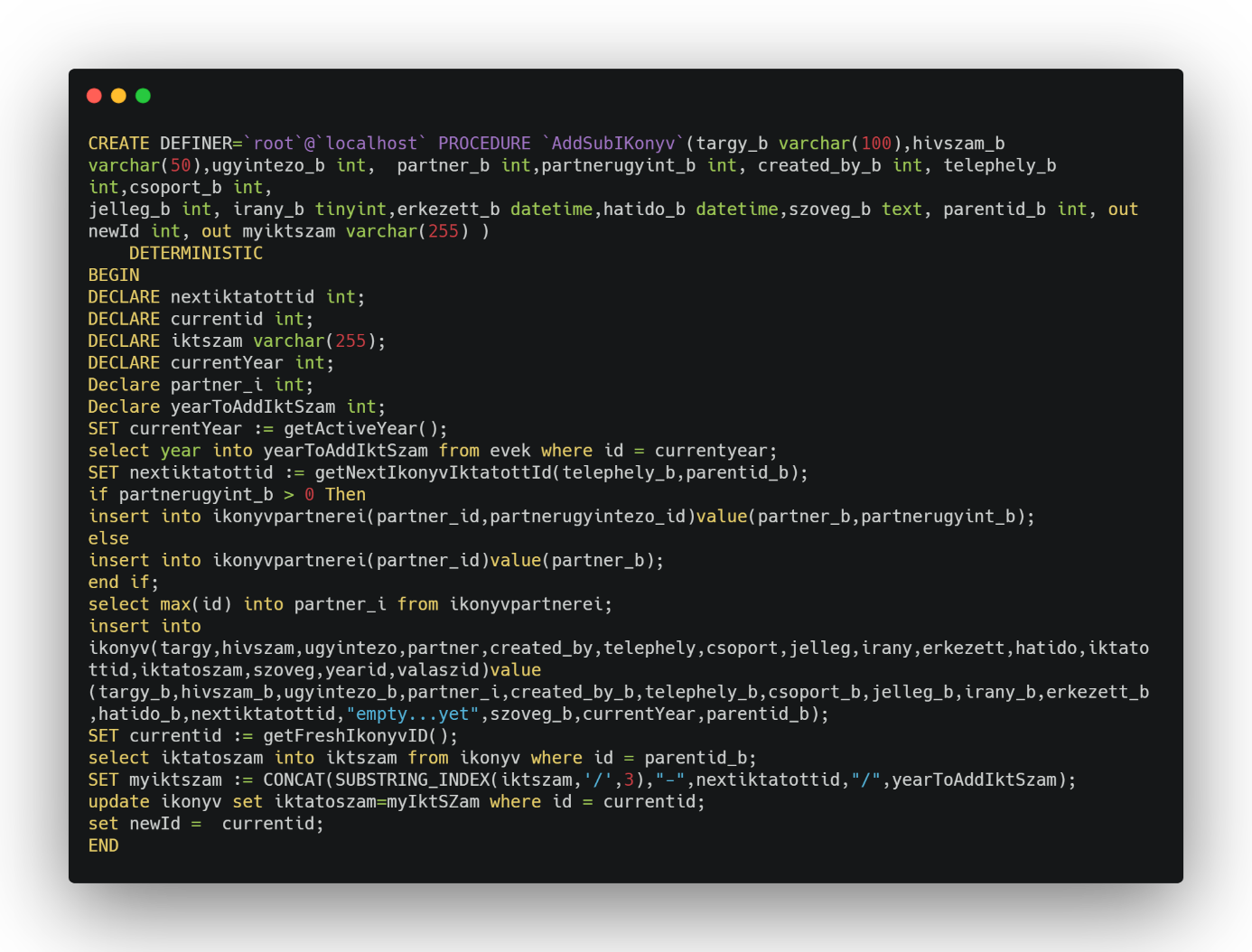
AddRootIkonyv, AddSubIkonyv, DelIkonyv, setDeletedByValaszID, getNextIktatottID, getIkonyvek. Ezek okozták a legtöbb fejtörést a számomra.

* + 1. **AddRootIkonyv:**

****

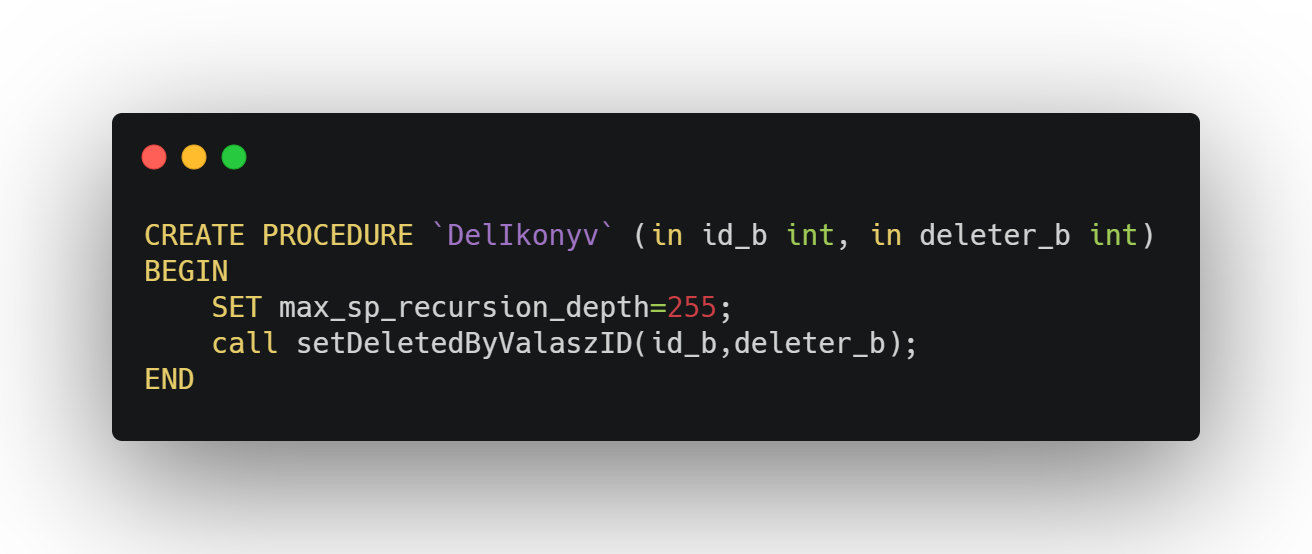
Bemenete az iktatóhoz szükséges adatok mint a Tárgy, Hivatkozási szám, Ügyintéző id-je, partner, partnerügyintéző id, felhasználó id, telephely, csoport, jelleg,irany id-k és persze a határidő és a érkezett dátum, megjegyzés szövege. Kimenete az új id és az új iktatószám lesz, mivel a többi dolgot a 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.

* + 1. **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 az szülő iktatószámát és azt átalakítva mentem el az ikönyv táblában.

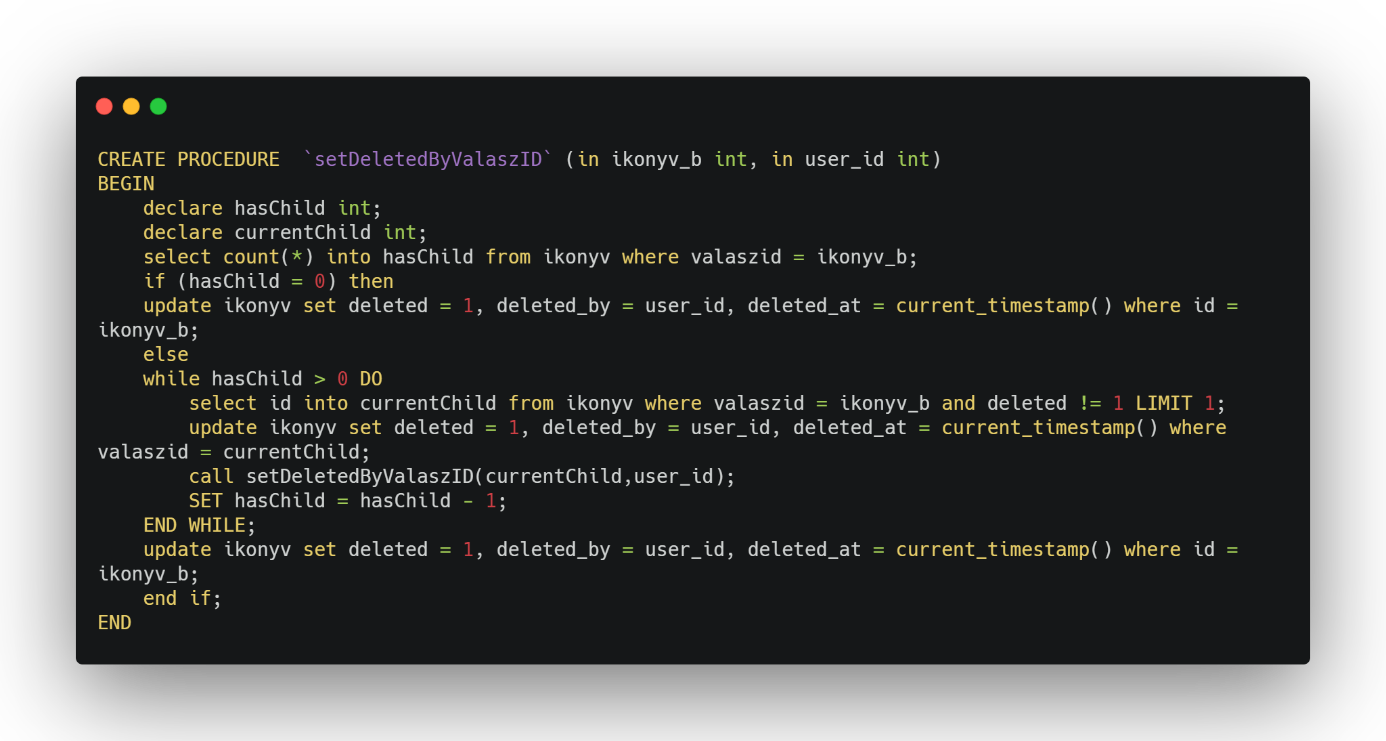
* + 1. **DelIkonyv:**



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 meghivja a setDeletedByValaszID-t a bemenő paraméterekkel.

* + 1. **setDeletedByValaszID:**

****

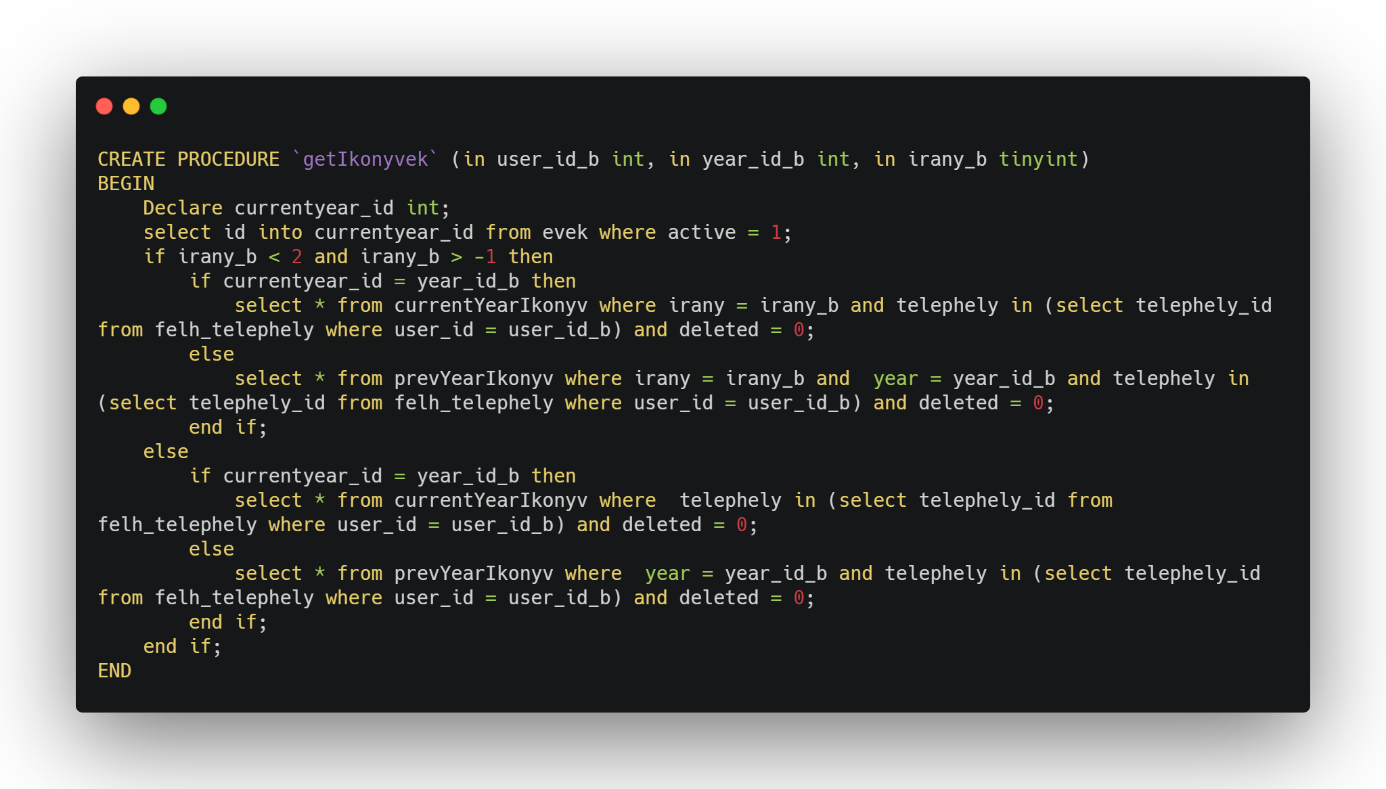
A bemenő paraméterek ugyan azok mint a DelIkonyvnek. Első lépésben megnézzük, hogy az iktatásnak vannak-e gyermekei, ha nincs akkor a flaget 1-re állítjuk ha van akkor a következő lépéseket kezdjük. Amíg van gyermeke addig kiválasztjuk az első gyermeket. A gyermek delete flagjét 1-re állítjuk és meghívjuk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

* + 1. **getNextIktatottID:**

****

A bemenete a telephelynek az id-je és hogy ha van akkor a szülőnek az id-je. Először meg nézem 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.

* + 1. **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. Illetve a lapozáshoz még fog kelleni egy tól-ig paraméter is.

* 1. **Nézetek:**

Egyelőre három nézetem van. Első a creatIktSzam ahol a iktató könyv tábla összes idegen kulcsa össze van kapcsolva a táblájával és ezekből összeállítja az aktuális iktató könyv iktatószámát. A második a 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.

1. **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:

*8.1. 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.

*8.2. 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.

*8.3. 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.

8.4. *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.

1. **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 egy helyen 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.