DIPLOMAMUNKA

Dubravszky Csaba Pál

Debrecen
2010

Debreceni Egyetem

Informatika Kar

SZOFTVERFEJLESZTÉS LOTUS NOTES KÖRNYEZETBEN

Témavezető:
Dr. Zichar Marianna
Egyetemi adjunktus

Készítette:
Dubravszky Csaba Pál
Programtervező matematikus

Külső konzulens:
Csörgő József
National Instruments

Debrecen
2010

Tartalomjegyzék

1. Bevezetés

Több nagyvállalat is használja a Lotus Domino / Notes rendszert, ezért témaválasztásom nem titkolt célja, hogy a megszerzett tudásom esetleg előnyt jelentsen a későbbi munkahelykeresés során.

Diplomamunkám célja, mint azt címe is mutatja, az hogy bemutassam a Lotus Notes fejlesztői eszközeit egy általam készített alkalmazáson keresztül. Az alkalmazás fejlesztése során felhasználtam a Notes rendszerben megjelent új technológiát, az XPagest, amire ki fogok térni. Nem célom ellenben a teljes Lotus Notes részletes bemutatása, mivel erre több száz oldal is kevés lenne. Az összetettebb bonyolultabb kódokat sem kívánom itt részletezni, tekintettel arra, hogy nem a Notes által használt programnyelvek ismertetése a cél. A főbb eszközöket kívánom nagyvonalakban bemutatni, részletesebben pedig majd az alkalmazás bemutatásánál beszélek róluk. Így egy konkrét példán keresztül elmagyarázni, nem a száraz elméletre, hanem inkább az érdekesebb és kézzel foghatóbb gyakorlati részre helyezve a hangsúlyt.

1.1. A Lotus Notes története röviden

A Lotus Notes gyökerei az Illinois Egyetem Számítógépes Oktatási Kutatóközpontjának (Computer-based Education Research Laboratory - CERL) első számítógépes programjai között találhatóak meg. 1973-ban a CERL piacra dobott egy PLATO Notes nevű terméket. Ennek mindössze annyi feladata volt, hogy egy hibajelentéshez hozzáfűzte a felhasználó azonosítóját, az adminisztrátorok pedig elküldhették rá válaszukat.

1976-ban kiadták a PLATO Group Notesot, amely a PLATO Notesot bővítette ki csoportmunkára alkalmassá. Népszerűsége megmaradt a 80-as évek elejéig. Majd megjelent az IBM első személyi számítógépe és 1982-ben az MS-DOS. Ray Ozzie aki PLATO rendszeren dolgozott a késő 70-es években, elkezdett dolgozni egy PC alapú Notes verzión is, de nem talált támogatóra eleinte. De Mitch Kapor, a Lotus Development Corporation alapítója lehetőséget látott a Notesban és befektetett. 1984-ben létrejött az Iris Associates Inc. amelyet a Lotus támogatott és célja a Lotus Notes első kiadásának kifejlesztése volt. Az elején nehézségekkel kellett szembenézniük, mivel az elképzeléseik szerint Notes online vitafórumokat, e-mail szolgáltatást, telefonkönyveket és dokumentumtárakat tartalmazott volna, de a hálózati munka akkoriban még nem volt úgy elterjedve, és az operációs

rendszerek se voltak elég érettek hozzá, sok rendszerszintű programozásra volt szükség. Majd ahogy a hálózati munka jelentősége nőtt, a Notesot úgy kezdték árulni, mint a csoportmunka egy eszköze. A Notes messze megelőzte a saját korát, és még a mai napig sem érte el más a Notes minőségét. 1986-ban a termék, szállításra készen állt a belső felhasználóknak, 1987-ben a Lotus megvette a jogokat a Notesra. Az első kiadást 1989-ben szállították. A Notes folyamatosan fejlődött, és bővült. 1995-ben az IBM megvásárolta a Lotust, leginkább a technológia megszerzése miatt. Az IBM anyagi háttere jót tett a Notes fejlődésének. Azóta is folyamatosan fejlődik és sikeres. Jelenleg a 8.5-ös verziónál járnak. [2]

1.2. A Lotus Notesról általánosságban

Arra a kérdésre, hogy mi a Lotus Notes, sok embertől hallottam már azt a választ, hogy egy levelezőrendszer. Ez pedig így nem helytálló. Igaz, hogy a kommunikáció nagy szerepet kap benne, mint hogy egy csoportmunkát segítő szoftver, de nem csak egy levelező. Az IBM oldalán a következő rövid kis leírás található: „Együttműködési szoftver az információk hatékony megosztásához és menedzseléséhez, az üzleti döntések felgyorsításának és a feladatok zökkenőmentesebb végrehajtásának érdekében”. Tehát ez egy együttműködést segítő szoftver, amely segíti a kapcsolattartást és kis alkalmazásokon, avagy adatbázisokon keresztül az információk kezelését. Természetesen a Notesnak is megvannak a hátárai. Nagyon nagy és nagyon sok, illetve nagyon összetett adatok kezelése már problémát jelenthet, nem egy Oracle, és nem is relációs adatbázis kezelő, de nem is erre lett szánva. Ám ha nem a határait feszegetve használjuk, akkor ez egy kényelmes és jól használható csoportmunka szoftver. Nem hiába sikeres és képvisel egyedüli minőséget ezen a piacon.

2. Design elemek

2.1. Form

A form, magyarul űrlap, azaz eszköz, aminek segítségével létre tudjuk hozni dokumentumainkat. Ez a dokumentum sablonja, a struktúrát biztosítja az adatbevitelhez, valamint a megjelenítéshez. Három típusa létezik:

- Dokumentum típus: Ez a típus a legfelső szinten áll, hozzá tartozhatnak választípusúak
- Választípus: Ezzel a típussal egy fődokumentumhoz tartozó válaszdokumentumot hozhatunk létre. Ez a nézetekben a fődokumentuma alatt és kissé behúzva jelenik meg.
- Válasz a válaszra típus: Ennek segítségével egy fődokumentumhoz tartozó válaszdokumentumot vagy egy válaszdokumentumhoz tartozó válaszdokumentumot hozhatunk létre.

A válaszoknak 32 szintjét tudjuk megjeleníteni.

A formnak megadható alias név is úgy, hogy amikor megadjuk a nevét utána egy | (pipe) karakter után írjuk az alias nevet. Ennek akkor lehet jelentősége, ha programozásnál hivatkozunk a form nevére.

Amikor létrehozunk egy új formot akkor gyakorlatilag egy szövegszerkesztő ablakot kapunk, amibe írhatunk statikus szövegeket, beszúrhatunk táblázatot vagy akár képeket is, formázhatjuk a szöveget, mint egy szokványos szövegszerkesztőben. A formhoz tartoznak még események melyekhez a kódot több nyelven is megírhatjuk, de ezeket majd később ismertetem. Az eseményekkel gyakorlatilag a funkcionalitások programozhatóak.

2.1.1. Mező

Eddig láthattuk, hogy el tudunk helyezni statikus szövegeket és objektumokat a formon, de mi tárolja azt az információt, ami a formmal létrehozott dokumentumonként más és más? Erre szolgálnak a mezők. A mezők szintén különböző típusúak lehetnek, mint például: szöveg, szám, rádiógomb, idő, stb. Összesen 17 típus létezik, melyeknek sok egyéb tulajdonsága is beállíthatók. A legfontosabb tulajdonságnál, mely szintén a típushoz kötődik, és közvetlenül mellette is kell megadni négy lehetséges érték közül választhatunk:

- Szerkeszthető: Nevéből is adódik, hogy ekkor a mező tartalma szerkeszthető
- Számított: Nem szerkeszthető a mező tartalma, az értéke valamilyen általunk magadott képlet alapján számítódik ki, és bizonyos események esetén -pl. dokumentum mentésekor- újraszámítódik.
- Létrehozáskor számított: Olyan, mint a számított, azzal a különbséggel, hogy létrehozásnál kiszámítódik az értéke, és utána nem változik akkor sem, ha módosítjuk a dokumentumot.
- Megjelenítéshez számított: Értéke nincs eltárolva a dokumentumban, csak akkor számítódik ki mikor a dokumentumot megnyitjuk, bezáráskor pedig értéke elveszik.

Bár a számított mezőknek közvetlenül nem szerkeszthető a tartalma, valamely programozási lehetőséget használva viszont szerkeszthető egy back-end dokumentum mezőjének tartalma. A mező szerkeszthetőségére vonatkozó megszorítás csak a front-end dokumentumokra érvényes. A back-end és front-end dokumentumok fogalmáról a programozási lehetőségek fejezetben lesz szó.

A mező nevére vonatkozó korlátozások:

- Egy formon a mező nevének egyedinek kell lenni.
- Hossza maximum 32 karakter lehet
- Nem kezdődhet számmal, dollár illetve kukac karakterrel. Szóközt nem tartalmazhat, ékezeteket viszont igen. Nem ajánlott azonban az ékezetek használata mivel a LotusScript programozási nyelv nem tudja kezelni ezeket. A konvenció az, hogy a mezőnek első betűje nagybetű, a többi kisbetű és ékezeteket nem tartalmazhatnak.
- Néhány mezőnév már foglalt a Notesban, ezek speciális jelentéssel bírnak Ezek a mezőnevek a következők: Form, SendTo, CopyTo, Sign, EncryptSaveOptions, MailFormat, DeliveryPriority, Categories

A mező akárcsak a form, rendelkezik eseményekkel.
Röviden összefoglalva egy grafikus felületen kényelmesen rakhatjuk össze a dokumentum létrehozásához és megjelenítéséhez szükséges formunkat és programozhatjuk a funkcionalitásokat a különböző eseményeken keresztül.

2.2. Nézet

A nézet azon eszköz amiben megjelennek a dokumentumaink. Sorai egy-egy dokumentumot tartalmaznak, az oszlopai pedig valamilyen információt a dokumentumból. Ez lehet egy mezőnév, vagy lehet egy képlettel kiszámított érték is. Amikor létrehozunk egy nézetet, ahhoz, hogy lássuk is a dokumentumainkat, a következő dolgok megadása szükséges: Meg kell adni egy feltételt, ami szerint kiválasztja a dokumentumokat. Ezt megtehetjük egyszerű művelet segítségével, vagy képlet segítségével. Képlet segítségével bonyolultabb feltételeket is írhatunk. Egy nézetben több különböző formmal készült dokumentum is megjeleníthető, arra azonban figyeljünk oda, hogy ez nem relációs adatbázis-kezelő. Hiába egy form tartalmazza a struktúrát, mint ahogy a relációs adatbázisnál a tábla, és adná magát a párhuzam, nem alkalmazhatjuk két form között az összekapcsolás műveletét, mivel itt ez a művelet nem létezik. Emiatt gyakran szükség lehet redundáns adattárolásra, vagy pedig formjainkat és nézeteinket úgy kell megtervezni, ha lehetséges, hogy a felhasználó mégis könnyedén megtalálja amit keres. Megadható a nézet típusa is, leggyakrabban az osztott nézeteket használjuk. Ezek mindenki számára hozzáférhetők, az adatbázis tervezője hozza létre őket. Valamint vannak privát nézetek, amit a felhasználó hoz létre, és csak ő láthatja.
Miután létrehoztuk nézetünket, definiálni kell az oszlopokat benne. Mint fentebb említettem egy oszlop megjeleníthet egy mezőt, vagy egy képlet eredményét is. Így akár egy oszlopban megjeleníthetjük több mező tartalmát is, illetve egy vagy több mező tartalmán végzett műveletet eredményét is. Itt jegyzem meg, hogy Rich Text tipusú mezőt nem lehet nézetben megjeleníteni. Az oszlopok tulajdonságai között természetesen beállítható a rendezés is. A Notes a rendezettséget mindig balról jobbra értelmezi, vagyis ha az első oszlop tartalma ugyanaz, csak akkor rendez a második oszlop szerint. [1]

2.2.1. Kategorizált nézet

Ha több olyan sorunk is van ahol egy oszlop tartalma ugyanaz, akkor érdemes kategorizált nézetet csinálni. Például egy CD nyilvántartó esetén, ahol az első oszlop az előadó, a második a dal címe. Ebben az esetben kategorizálhatjuk előadó szerint. Ehhez nem kell mást tenni, mint azt, hogy az első oszlop tulajdonságainál beállítjuk, hogy a típusa kategorizált legyen. Ekkor az előadó oszlopban egyszer fog szerepelni az előadó neve, és alatta a második oszlopban lesznek a számcímek. A kategóriát össze is lehet csukni, illetve kinyitni, ha kétszer kattintunk a nevére. A kategorizált oszlop elé rakhatunk még egy kis

háromszöget, amire egyet kattintva nyitható és zárható a kategória. Ezt a „Show twistie when row is expandeble” opcióval tehetjük meg az oszlop tulajdonságinál.
Így ha van öt előadónk, és mindegyiknek tíz dala, akkor nem 50 sort fogunk látni hirtelen, csak ötöt, és csak azt nyitjuk ki, akinek a dalai érdekelnek.

2.2.2. Válaszdokumentumok megjelenítése a nézetben

1. A nézet tulajdonságaiban be kell állítani, hogy a válaszdokumentumok hierarchia szerint jelenjenek meg:„Show response documents in hierarchy” Ez azért szükséges, hogy behúzva jelenjenek meg ezek a dokumentumok a fődokumentum alatt. Persze nem kötelező, ha nem akarjuk, így látni őket.

2. Szükségünk lesz egy speciális oszlopra, ami csak a válaszdokumentumokat jeleníti meg. Ezt az oszlopot általában legelőre tesszük és nincs neve. Ehhez ennek az oszlopnak a tulajdonságainál be kell állítani, hogy csak a válaszokat mutassa („show responses only”), valamint érdemes beállítani itt is, a kis háromszög mutatását, ha a sor összecsukható, mint a kategorizált nézetek esetén. Mivel itt egy oszlopban kell megjelenítenünk a dokumentum minden mezőjét, ezért csak képlettel adhatjuk meg, amit meg szeretnénk jeleníteni.

2.2.3. Nézet csoportok

Az adatbázis dokumentainak elérésére szolgáló nézetekből csoportokat lehet kialakítani. Gyakorlatilag a mappák hierarchikus elrendezéséhez hasonlóan kialakítható a nézetek hierarchiája is. Ez egyrészt lehetőséget ad a logikai csoportosításra, másrészt jól használható, ha egy nézet egy másik leszűkítéseként jön létre. A nézet definiálásakor kiválszthatjuk a már létező hierarchiából az új nézet pontos helyét. Ezt később a nézet megfelelő átnevezésével megváltoztatható: nézet csoport név\nézet név.

2.2.4. Szerviz nézet

Ez olyan nézet, amit csak a Designerből lehet látni, a Notes kliensből nem, el van rejtve a felhasználók elől. Akkor használjuk, ha szükségünk van egy külön nézetre, például egy keresés miatt, de más szerepet nem játszik és ezért nem is szükséges megjeleníteni. Ilyen nézetet úgy hozhatunk létre, hogy a nevét zárójelbe tesszük.

Nézeteknél szintén megadható alias név, ám itt nem kell olyan „trükkösen” megadni mint a form esetén, ugyanis a nézeteknél külön van egy alias név mező.

2.3. Művelet

A műveletek (action) valamilyen általunk meghatározott műveletet hajtanak végre. Leginkább gomb formájában nyilvánulnak meg. Létrehozhatók formokon és nézeteken is egyaránt. A végrehajtandó műveletet megadhatjuk egyszerű művelet segítségével, vagy ha összetettebbre van szükségünk, megírhatjuk, amire a Notes több nyelven is lehetőséget biztosít.

2.4. Gyűjtő

Ez egy speciális nézet, melybe a dokumentumok nem egy szelekciós feltételnek való megfelelés következtében kerülnek be, hanem maga a felhasználó döntheti el, hogy mely dokumentumok legyenek elérhetőek az adott gyűjtőből. Jellemző példa a kedvenc dokumentumok kigyűjtése egy gyűjtőbe.

2.5. Ügynök

Az ügynökök gyakorlatilag kis programok, amelyek valamilyen tevékenységet hajtanak végre. Egy ügynök futása lehet ütemezett, vagy köthetjük valamilyen adatbázis esemény bekövetkezéséhez, vagy akár manuálisan is futtathatjuk őket. Az esemény hatására elindulókat, az Oracle triggerekhez tudnám hasonlítani, amelyek szintén valamilyen esemény hatására futnak le.Egy ügynök lehet:
Osztott: mindenki számára elérhető és használható, az adatbázis tervezője hozza létre.
Személyes: A felhasználó saját maga hozza létre, és csak ő láthatja.

Az ügynök törzse lehet: formulák, egyszerű művelet, LotusScript, JavaScript, imported Java.
A formulák és az egyszerű műveletek minden dokumentumon lefutnak egyesével, amire az ügynök hatása kiterjed. A LotusScript és a JavaScript viszont csak egyszer fut le, és itt nekünk kell úgy megírni a kis programot, hogy az mindegyik dokumentumon lefusson. Ha az egyszeri futtatást választjuk, és formulával határozzuk meg az ügynök tevékenységét, akkor csak egyszer lesz végrehajtva, nem minden dokumentumon külön-külön.[1]

Egy ügynököt van lehetőségünk a kliensen, illetve szerveren futtatni. Utóbbi a praktikusabb, mivel ha a kliensen fut az ügynök meg kell várnunk a lefutását , addig nem használhatjuk a kliensünket. Szerveren futtatás esetén több szervert is be lehet állítani ahol az ügynököt tartalmazó adatbázis megtalálható, ám ezzel óvatosan kell bánni, mert ha az ügynök megváltoztat dokumentumokat és ezek az adatbázisok egymással replikálódnak akkor az replikációs konfliktushoz vezethet. Ilyenkor érdemesebb egy konkrét szervert beállítani, és a replikációt úgy, hogy az ügynök futása után történjen meg. Valamint fontos még tudni, ha egy szerveren ütemezett ügynököt akarunk futtatni ahhoz megfelelő jogokkal kell rendelkezünk. Ezt a szerverdokumentumban az ügynökkorlátozások részen állíthatjuk be.
Az ütemezett ügynökök szerveren való futtatásakor vegyük figyelembe, hogy bár az ügynök futásához az kell, hogy a létrehozójának legyenek megfelelő jogai a futtatásra, mégis a futtatáskor a szerver a szerver ID-vel futtatja őket. Tehát ahhoz, hogy az ügynök meg tudjon változtatni dokumentumokat, az adatbázisra, ill. az egyes dokumentumokra a szervernek biztosítani kell az írási, olvasási stb. jogot.[1]

2.6. További fejlesztői eszközök

Subform

Jelentősége abban áll, hogy egy ilyen subformot (részűrlapot) beszúrhatunk több formra is. Így azon elemeket, amik több formon is megjelennek elég csak egyszer elkészíteni.

Osztott mező

Szintén az újrafelhasználást segíti. Olyan mezők, amik több formon is előfordulnak azonos tulajdonságokkal, elég egyszer egy helyen létrehozni. Azonban vigyázni kell, ha már használatban vannak és meg akarjuk változtatni őket.

Osztott oszlop

Célja megegyezik az előzőekkel, nézetek esetén használhatjuk őket.

Osztott művelet

Egy helyen elég létrehozni a műveleteinket, majd aztán több helyen is felhasználhatjuk őket. Rátehetjük őket nézetekre, formokra.

Navigátor

Ennek az eszköznek nem a funkcionalitásban, inkább csak a külcsínben van szerepe. Használata nélkül is, tökéletesen működő alkalmazás hozható létre. Használatukkal szépíthetjük az alkalmazásunk megjelenését, bár létrehozásuk kicsit sziszifuszi munkával jár ha azt szeretnénk, hogy tényleg úgy nézzen ki ahogy mi akarjuk.

Forrópontok (Hotspot)

Használatukkal segítséget adhatunk a felhasználóknak a mezők kitöltéséhez, illetve valamilyen műveletet hajthatunk végre. Öt típusa van: felbukkanó szöveg, felbukkanó kiszámított szöveg, gomb, műveleti forrópont, lánc forrópont.

Az első kettő használható a felhasználó segítéséhez. **Felbukkanó vagy felbukkanó kiszámított szöveg** típusú forrópontot hozzáadhatunk egy szöveghez formon amire, ha a felhasználó rámutat, vagy rákattint, megjelenik az általunk definiált felbukkanó szöveg.

Gomb esetén egy gombot hozhatunk létre a formon, amellyel valamilyen általunk megírt műveletet hajthatunk végre.

A **műveleti forrópont** szintén egy műveletet hajt végre, ám ez nem egy gomb lesz, hanem az általunk kijelölt szöveghez adhatjuk hozzá, így a szövegre kattintva végrehajtódik az általunk meghatározott művelet.

Lánc forrópontot szintén egy kijelölt szöveghez tudjuk kötni, segítségével megnyithatunk egy oldalt, dokumentumot, vagy megadhatunk egy tetszőleges URL-t.

3. A programozási környezet

Mint már említettem a Notes több nyelven is lehetőséget ad nekünk a programozásra. Formulanyelv (képlet), LotusScript, JavaScript, Java. Bizonyos helyeken választhatunk bármelyik közül, hogy melyiket kívánjuk használni, máshol pedig csak kevesebb közül, vagy egyáltalán nem választhatunk, a Notes dönti el, hogy melyiket lehet használni.

Az eseményeket lehet programozni. Ezek lehetnek adatbázis események, lehetnek mezőn, vagy a formon végbemenő események, vagy lehetnek egy gomb megnyomása folyamán kiváltódó események. A kis programjainkat megírhatjuk azon a helyen, ahol az esemény el van helyezve, vagy elhelyezhetjük őket szkript könyvtárakban és majd a megfelelő helyről meghívjuk őket. Ez jó abból a szempontból, hogy a kódjaink egy helyen helyezkednek el, de

leginkább akkor hasznos, ha egy adott kis programot több helyről is meg kell hívnunk, így elég csak egyszer megírni egy helyen.

3.1. Programozási lehetőségek

Egyszerű művelet

Nem kíván programozást, csupán egy ablak tartalmát kell értelemszerűen kitölteni, legördülő listából választhatunk, milyen műveletet hajtson végre a Notes. Nagyon alapvető dolgokat lehet így megtenni.

Formula nyelv (Képlet)

Itt már bővülnek a lehetőségeink, összetettebb műveleteket is megvalósíthatunk, de ez még mindig inkább alacsonyabb szintű műveletek végrehajtását jelenti. Viszont ezekben az esetekben egyszerűen használhatóak, és gyorsan futnak.
Egy ilyen képletet vagy formulát a következő elemekből állíthatunk össze: változók, állandók, műveletek, fenntartott kulcsszavak, @függvények, @parancsok.
Nem kívánom részletezni használatát, de megemlítenék két nézetekben gyakran használt @függvényt, amelyek létezéséről jó tudni:

A *@DbColumn* függvény megkeresi és előhívja az adott adatbázis egyik nézetének, vagy gyűjtőjének adott oszlopában található összes értéket. A függvény tehát egy listát ad vissza.

A *@DbLookup* függvény megkeresi az adott adatbázis egyik nézetének, vagy gyűjtőjének első oszlopában a megadott kulcs szerinti értéket tartalmazó dokumentumot vagy dokumentumokat. A függvény a dokumentum(ok)nak a megadott nézet és oszlopszámban található vagy az adott mezőnevű értékét adja vissza. Ez a függvény tehát dokumentumonként egy értéket ad vissza. [1]

Ez a két függvény oszlopképletben valamint mező értékének számításakor használtható. Kiválasztási képletben nem. Erre figyeljünk oda, hogy melyik képlet milyen helyeken használható, ezért figyelmesen olvassuk el a leírásukat, bármennyire is triviálisnak tűnik az a függvény a neve alapján. Ugyanis nem minden függvény használható mindenhol, és nem

megfelelő helyen használva őket nem várt eredményt kaphatunk, vagy nem is kapunk eredményt.

LotusScript

A LotusScript adja meg számunkra a legnagyobb programozói szabadságot a Notesban. Ez egy objektum alapú Basic-szerű nyelv. Használata bonyolultabb, mint az előzőek, ám nagyon sok mindent megtehetünk benne, amit az előzőekkel nem. Mivel a LotusScript objektum alapú ezért tartalmaz előre definiált osztályokat. Ezen osztályok két fő részre oszthatók:
A back-end (adatbázis szintű) osztályok a Notes adatbázisaihoz, nézeteihez, dokumentumaihoz, ill. egyéb elemeihez való hozzáférést segítik.
A front-end (UI - User Interface, felhasználói felület) osztályok a felhasználó által éppen használt objektumokhoz való hozzáférést teszik lehetővé. Ilyen lehet, pl. az éppen nyitott dokumentum, nézet, adatbázis, stb.

Fontos megjegyezni, hogy a fenti tulajdonságok miatt a szerver által futtatott scriptek (pl. a szerveren futó ügynökök) csak back-end osztályokat, míg a felhasználó munkaállomása mindkettőt használhatja.[1]

Java script

Objektumorientált programozási nyelv. Leginkább a webes vagy webről és Notesból egyaránt használható alkalmazások esetén használjuk.

Java

Objektumorientált. Írhatunk natív Java kódot, de importálhatunk is Java applet fájlokat az alkalmazásba. Ügynököknél használható csak.

4. XPages

Az XPages egy új technológia a Notesban, a webes felületek kialakítására, amely a 8.5-ös verzióban jelent meg először. A Sun által kifejlesztett JSF (Java Server Faces) technológián alapulva lehetővé teszi 2.0-ás webalkalmazások létrehozását. Segítségével egyszerűbben hozhatunk létre weboldalakat, amelyek szebbek és funkcionalitásban is többet nyújthatnak mint az eddig Notesban létrehozottak. Nincs szükség külön létrehozni page-eket, frameseteket és outline-okat, hogy kialakíthassunk egy jó webes felületet, csak ezeket az úgynevezett XPage-eket. Ennek következményeképp nem is szándékozom bemutatni az XPages megjelenése előtt webes fejlesztésre használt eszközöket, mivel az alkalmazásom webes felületét szintén csak az XPages segítségével alakítottam ki.

4.1. A használatáról röviden

Mint ahogy a form létrehozásánál láthattuk, egy új XPage létrehozásnál is egy üres lapot kapunk egy szövegszerkesztővel, amin elhelyezhetünk statikus objektumokat, valamint vezérlőket. A megjelenítés formázásához nincs szükségünk HTML ismeretére, valamint JavaScript ismeretére se az egyszerűbb funkcionalitások megadásához. Bonyolultabbaknál viszont már szükséges lehet. A statikus objektumainkat és a vezérlőinket csak rá kell „húzni” az oldalra egy grafikus felületen, és beállítgatni a tulajdonságaikat. Az oldal kódja ezekből generálódik, de szükség esetén átválthatunk, hogy a kódot mutassa, és ott is módosíthatunk. A vezérlők segítségével tudunk adatokat megjeleníteni a Notesból illetve adatokat felvinni oda, vagy valamilyen tevékenységet végrehajtani az oldalon. Egyes vezérlőket, hogy használhatóak is legyenek, hozzá kell őket rendelni valamilyen adatforráshoz, vagy speciális változóhoz. Az adatforrás attól függően, hogy mire van szüksége az eszköznek több fajta is lehet : nézet, form, mező , konkrét dokumentum, dokumentum kollekció. Az adatforrás megadása is többféle módon történhet. Tehetjük azt egyszerűen legördülő listából választva, vagy Java script segítségével, vagy speciális változókat, úgynevezett hatásköri változókat rendelhetünk hozzájuk. Tehát az adott vezérlőt amely adatforrást igényel, hozzá kell rendelni valamilyen adatforráshoz, vagy változóhoz különben csak dísz lesz az oldalunkon. Ezenfelül minden vezérlő rengeteg tulajdonsággal rendelkezik még, és minden vezérlő rendelkezik eseményekkel, pontosabban mindegyiken kiváltódhat valamilyen esemény, szintúgy magán az XPage-en is. Ezekről bővebben az alkalmazás bemutatásánál beszélek.

5. A biztonságról

Adatainkat többféle módon is védhetjük az illetéktelenek elől. Háromféle védelem létezik: adatbázis szintű, dokumentum szintű és mező szintű védelem.

Adatbázis szintű védelem:

Ez a legáltalánosabb módja a védelemnek, melyet az adatbázis hozzáférési listáján (ACL) szabályozhatjuk. A felhasználóknak a következő hozzáférési szintek adhatók:

Nincs hozzáférés (No Access): Ilyenkor az adatbázishoz nincs hozzáférése.

Elhelyező (Despositor): Ilyenkor a felhasználó létre tud hozni dokumentumokat, de se a saját, se más a más által létrehozott dokumentumokat nem láthatja.

Olvasó (Reader): Az ilyen jogosultsággal rendelkező felhasználó az adatbázis összes dokumentumát olvashatja, viszont nem hozhat létre dokumentumot.

Szerző (Author): Ekkor az adatbázis összes dokumentuma olvasható valamint létre is lehet hozni saját dokumentumokat, de csak a sajátok szerkeszthetőek.

Szerkesztő (Editor): Létrehozhatók új dokumentumok, valamint olvasható és szerkeszthető az adatbázis összes dokumentuma.

Tervező (Designer): Az ilyen jogosultsággal rendelkező az adatbázisok szerkezetét, felépítését is meg tudja változtatni.

Kezelő (Manager): Aki ezzel a jogosultsággal rendelkezik az ACL-t is képes megváltoztatni.

Ezeket a hozzáférési szinteket még különböző privilégiumok hozzáadásával és elvételével lehet finomítani. Valamint lehetőség van szerepkörök (roles) létrehozására, amiket szintén hozzárendelhetünk az egyes felhasználókhoz. Valójában, ha létrehozunk egy szerepkört és hozzáadjuk egy felhasználóhoz az csak egy címke lesz. A megfelelő helyen nekünk kell leprogramozni, hogy mit tehet meg az adott szerepkörrel rendelkező felhasználó.

Létezik két speciális felhasználó típus, a Default és az Anonymous. Ha az adott felhasználó nem szerepel az ACL-ben, sem egyénileg, sem valamely csoport tagjaként akkor a Default jogait kapja, amennyiben pedig Web felhasználó akkor az Anonymous jogait.

Dokumentum szintű védelem:

Létezik két speciális mezőtípus. A szerző (author) és az olvasó (reader). Amennyiben egy szerző típusú mezőt helyezünk el a formon, akkor ezzel a formmal készített dokumentumot csak az szerkesztheti, aki szerepel ebben a szerző típusú mezőben, és legalább Author hozzáférési szinttel rendelkezik. Illetve ha több ilyen mező van, akkor ezek valamelyikében. Régebbi verziókban még úgy volt, hogy rendelkezhetett a felhasználó akár Manager hozzáféréssel is, ha nem szerepelt a szerző típusú mezőben, akkor nem szerkeszthette a dokumentumot. Újabb verziókban már az editor és magasabb szintű hozzáféréssel rendelkezők szerkeszthetik akkor is ha nem szerepelnek ebben a mezőben.

Az olvasó típusú mező szerepe hasonló. Aki nem szerepel ebben a mezőben, az nem olvashatja a dokumentumot.

Mezőszintű védelem:

Ez a legerősebb védelemfajta, mely lehetőséget ad arra, hogy a mezőket kulccsal vagy kulcsokkal titkosítsuk. Erre azért lehet szükség, mert attól még, hogy valaki nem szerepel az olvasó mezőben, hozzáférhet az adott dokumentum mezőjének a tartalmához, ha egy nézetben jobb gombbal megnyitja a dokumentum tulajdonságait. Az ablak mezők nevű lapján ugyanis elérhető a mezőlista és a mezők tartalma is. Egy mező titkosításához a következőket kell tenni: Létre kell hozni egy kulcsot, amit a File->Security->Notes Data-> Documents helyen tehetünk meg. A kulcsot elküldhetjük akár levélben, akár fájlba írhatjuk és így más módon is eljutathatjuk a célszemélynek. Majd a titkosítani kívánt mező tulajdonságok ablakában a speciális fülön a biztonsági beállításoknál engedélyezzük a titkosítást a mezőre. Majd az form tulajdonságainál a biztonság fülön válasszuk ki a használni kívánt kulcsot vagy kulcsokat az alapértelmezett titkosítási kulcsok részben.

Ha több kulcsot is megadtunk a titkosításnál, akkor a hozzáféréshez elég az is, ha csak az egyik kulccsal rendelkezik a felhasználó. Régebbi verziókban web felőli elrejtésre nem volt alkalmas, mivel onnan akkor is látszódtak, ha titkosított volt, de a 8.5-ös verzió súgója szerint már nem látszanak.

6. Az alkalmazás

A Notes funkcionalitásának szemléltetésére egy jegyzetkereső adatbázist fejlesztettem ki, mellyel, jegyzetek kereshetők illetve tölthetők fel. A jegyzeteknek megadhatjuk a tulajdonságaikat, a felhasználók írhatnak kommentárt egy jegyzethez, valamint pontozhatják is őket. Ilyen eseményekről a tulajdonos levélben kap értesítést, ha igényli. Valamint rendelkezik egy webes felülettel, így Notes kliens nélkül is elérhető az adatbázis, és a megjelenés is tetszetősebb bizonyos helyeken. A webes felületről nem lehet adminisztrációt végezni, mivel ezt feleslegesnek tartottam elkészíteni, hiszen az adminisztrátor úgyis rendelkezik klienssel, amiben bővebb lehetőségei vannak.

6.1. A Notes kliensbeli felület

6.1.1. Formok létrehozása

Az adatbázis öt formot tartalmaz. Ezek közül négy valósítja meg az alkalmazás logikáját, amely formok a következők: *Felhasznalo, Jegyzet, Komment, Beallitasok_profil*. Az ötödik a *Bongeszo* form pedig a webes hozzáféréssel kapcsolatos. (Az ékezetek szándékosan hiányoznak, mivel eszközök elnevezésekor nem használtam ékezetes betűket, ezért itt is így írom őket)

Felhasznalo form:

1. ábra

Ahhoz, hogy formunk az 1. ábrán láthatóan nézzen ki a designerben, a következőket kellett tenni. Létre kellett hozni egy táblázatot és oda elhelyezni a statikus szövegeket, majd pedig a mezőket. Ez eddig semmilyen különösebb tudást nem igényel, körülbelül szövegszerkesztés szintű dolog. Ez egy dokumentum típusú form lesz, tehát fődokumentumot hozunk létre vele.

Mezői:

A *Nev*, *E-mail*, *Telefon* mezők text típusúak és szerkeszthetők.

Az *Azonosito* author típusú és szerkeszthető. A mezőtulajdonságok control fülén beállíthatjuk, hogy a választás a Notes címjegyzékéből történjen és ne kellejen begépelni a felhasználó azonosítóját, hanem egy listából választhassunk. Ha ugyanitt bepipáljuk, hogy: „Lookup names for each character entered", akkor minden egyes billentyű lenyomás után megpróbálja kiegészíteni, ha még is úgy döntünk, hogy inkább begépelni akarjuk. A következő tulajdonságait töltjük ki ennek a mezőnek:

Default value: `""` Ez az alapértelmezett kezdőérték.

Input translation: `@Name([Canonicalize]; Azonosito)`

Ezzel a mezőbe bevitt értéket alakíthatjuk át. Itt az *Azonosito* mezőbe bekerülő azonosítót kanonikus alakra hozza.

Input validation: `@If(Azonosito="";@Failure("Az azonosítót kötelező megadni");@Success)`

Validálja a mező tartalmát. Ha a mező üres akkor hibaüzenetet ad.

Input enabled:

`@If(@IsMember("[uzemelteto]";@UserRoles);@True;@False)`

Csak akkor vihetünk be értéket a mezőbe ha rendelkezünk *uzemelteto* szerepkörrel. Erre azért van szükség, hogy aki nem üzemeltető az ne módosíthassa ezt a mezőt. Megoldható lett volna ez még úgy is, hogy a mezőt elrejtjük, ha ez a kifejezés igaz, és akkor sem lehet szerkeszteni. De én jobbnak tartom, ha mindig látszik. Az elrejtést a mezőtulajdonságok Hide when fülénél állíthatjuk be.

A *Honlap* rich text lite, szerkeszthető típusú. A mező tulajdonságaink control fülén csak a link van engedélyezve, hogy más csatolmányt ide ne lehessen betenni.

A *Tipus* radio button, szerkeszthető. Az input validation és input enabled tulajdonságok vannak csak kitöltve az előzőhöz hasonlóan. Három választási lehetőséget adunk meg szintén a mezőtulajdonságok control fülén a választási lehetőségeknél: „Oktató", „Hallgató", „Egyéb".

Az *Uzemelteto* check box , szerkeszthető. Csak az **input enabled** van megadva szintén a fenti módon. Választási lehetőségeknél egyet adunk meg: „Üzemeltető”.

A *LegIskVeg* combo box, szerkeszthető. Választási lehetőségek: „általános iskola”, „középiskola”, „főiskola”, „egyetem”.

A *LetrehozasiJog* check box, szerkeszthető. Választási lehetőségeknél megadjuk, hogy: „Létrehozhat dokumentumot“.

Default Value : Az egyetlen választási lehetőség, vagyis `"Létrehozhat dokumentumot"`. Így alapértelmezetten be lesz pipálva.

Input Translation: Ha az Üzemeltető be van pipálva, akkor ez is legyen bepipálva, egyébként meg értéke attól függjön, hogy bepipáltam vagy nem.

```
@If(Uzemelteto="Üzemeltető";"Létrehozhat
dokumentumot";LetrehozasiJog)
```

Input Enabled: Csak akkor lehessen változtatni az értékét, ha üzemeltető vagyok és az Üzemeltető nincs bepipálva.

```
@IsMember("[uzemelteto]";@UserRoles)&Uzemelteto!= "Üzemeltető"
```

A mező tulajdonságait így megadva azt értük el, hogy ha az Üzemeltetőt bepipáljuk, akkor a létrehozási jog is automatikusan be lesz pipálva, és nem lehet megváltoztatni. Ahhoz, hogy ez rendben működjön is, az *Uzemelteto* mező tulajdonságai között be kell állítani, hogy frissítse a mezőket, ha változik az értéke.

A *TorlesiJog* Checkbox, szerkeszthető. Egyetlen választási lehetőség: „Törölhet dokumentumot”. Ez lesz az **alapértelmezett értéke** is. Az **input enabled** tulajdonság pedig, ugyanaz, mint az *Uzemelteto* mezőnél, csak akkor változtathatja meg ha üzemeltető.

Érdemes a form tulajdonságainál még beállítani, hogy automatikusan frissítse a mezőket, így ha át akarjuk ugrani valamelyik kötelezően kitöltendő mezőt, akkor rögtön jelez, hogy töltsük ki. Valamint elhelyezzük a formon az alábbi szokásos műveleteket: *Ment és kilép*, *Szerkeszt, Kilép*, melyek kódját megírhatjuk itt is, de kényelmesebb osztott műveletként elkészíteni, mert ezeket minden formon elhelyezzük.

Forrópont:

Az *Azonosito* mező alatt elhelyezünk egy gomb típusú forrópontot, amelyet ha megnyomunk, akkor a begépelt részletből felajánl egy listát, hogy mi lehet a teljes név. Ezt a Notes címjegyzékéből keresi ki. Bár már beállítottuk a mező tulajdonságainál, hogy minden billentyűleütés után egészítse ki a nevet, de ez esetben látványosan kapunk egy listát, amiről választhatunk, ha több névére is kiegészíthető a begépelt részlet.

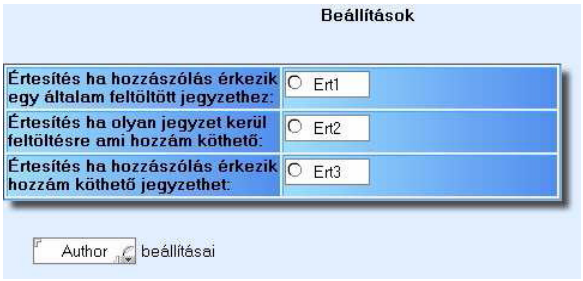
Eseményei:

A form tulajdonságainál az **initialize** és **terminate** eseményeket írjuk meg LotusScript nyelven, mégpedig úgy, hogy az inicializálásnál lekérdezzük, hogy új dokumentum-e, ezt eltároljuk, terminálásnál pedig ha új dokumentum, akkor meghívjuk a profildokumentumot létrehozó metódust az *Azonosito* mezőben található azonosítóval. Így, egyúttal mikor elkészítjük egy felhasználó adatlapját, akkor a profildokumentumát is létrehozzuk. A kódot itt nem részletezem, mindössze annyit jegyeznék meg, hogy az egyes eseményekben használt változók lokálisak. Globális változót, ami minden eseményéből látszik a formnak, a *Declarations* részben kell deklarálni. Ha pedig valamelyik szkript könyvtárunk függvényét akarjuk hívni akkor az *Options* részben kell használni a `Use "szkriptkönyvtárnév"` utasítást.

Valamint meg kell még írni a **Postsave** eseményt. Az itt megírt kód feladata az lesz, hogy amennyiben új felhasználóról van szó akkor felveszi az ACL-be is a megfelelő jogosultságokkal, amelyeket az *Uzemeleteto*, *LetrehozasiJog*, és *TorlesiJog* mezők értékei alapján határoz meg. Ha a felhasználót nem most hozzuk létre, csak szerkesztjük, akkor ez is csak szerkeszti az ACL-ben a jogosultságait.

Megjegyzés: Azt, hogy a dokumentum új-e, le lehetett volna kérdezni akár a Queryopen, illetve Querysave eseményeknél is. Viszont azt a metódust, amelyikkel létrehozom a profildokumentumot, már nem hívhatom meg a Postsave eseménynél, ahol esetleg inkább helye lett volna, mert hibás működést eredményez.

Beallitasok_profil form:



2. ábra

Mezői:

Az első három rádió gomb típusú mező melyek szerkeszthetőek. **Alapértelmezett értékük :** `"Nem"`.

Két választási lehetőség van megadva mindegyiknél: „Igen” valamint „Nem”.

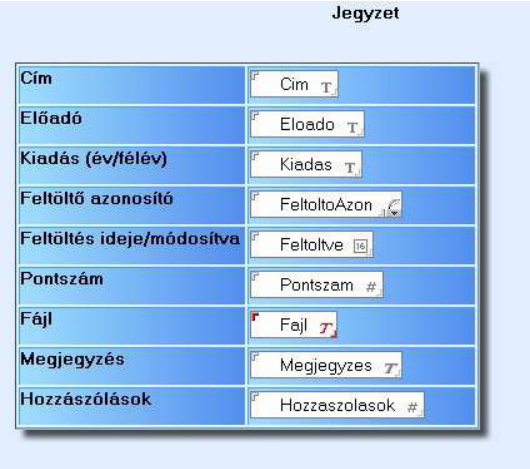
Az utolsó author típusú mező és szerkeszthető, amennyiben üzemeltető a szerkeszteni kívánó személy (**input enabled**) , valamint kötelező kitölteni (**input validation**).

Ez egy profil form lesz mellyel profil dokumentumokat tudunk létrehozni. A profil form semmiben nem különbözik egy szokványos formtól, egyedül annyi változtatást kell eszközölnünk, hogy nem jelenítjük meg a menüben. Ezt a form tulajdonság ablakának első fülén állíthatjuk be azzal, hogy kivesszük a pipát az „include in menu”-nél. Erre azért van szükség, mert a profil dokumentumokat nem a szokványos módon kell létrehozni a profil formmal. Ha így tennénk, akkor csak egy szokványos dokumentumot hoznánk létre. Profil dokumentumot egy külön erre szolgáló paranccsal tudunk létrehozni, illetve szerkesztésre megnyitni. Ez szintén megtehető több nyelven. Nem részletezném mindegyik nyelven a szintaktikát, csak általánosan írom le. Ez a parancs formula nyelvben az *EditProfile* vagy *EditProfileDocument*, LotusScript-ben *EditProfile* névre hallgat. Két argumentuma van: a profil form neve és egy egyedi kulcs. Ez utóbbi opcionális. A kulcs általában egy felhasználónév, így minden felhasználóhoz létrehozhatunk egy profil dokumentumot, amiben a felhasználók beállításait tárolhatjuk. Ha a kulcs elmarad, akkor egy adatbázis profilt hozunk létre, amiből csak egy lehet az adatbázisban. Itt tárolhatjuk például a globális, az egész adatbázisra vonatkozó beállításokat. A profil dokumentumok láthatatlanok, tehát egy nézetből sem láthatóak. Jelentőségük abban áll, hogy gyorsan és egyszerűen el lehet őket érni az egyedi kulcs segítségével.

Lehetőségünk van még arra is, hogy lekérdezzük, illetve megváltoztassuk egy profildokumentum mezőjének tartalmát anélkül, hogy megnyitnánk őket. Formulanyelven erre szolgálnak a *GetProfileField* és *SetProfileField* parancsok. Ha a profildokumentum nem létezik, akkor a *SetProfileField* létre is hozza. LotusScrip és JavaScript nyelven pedig a *GetProfileDocument* metódussal tudunk kikérni egy profildokumentumot, majd lekérdezni

vagy megváltozatni mezőit. Ha a profil dokumentum eddig nem létezett, ez metódus szintén létre hozza. A profil formon szükséges még elhelyezni egy author típusú mezőt, amibe szerepelnie kell az adott felhasználónak, hogy tudja is szerkeszteni a saját profildokumentumát.

Jegyzet form:



3. ábra

Ez a form szintén dokumentum típusú.

Nem részletezem külön, hogy mely tulajdonságait adjuk meg egy mezőnek, mivel az előző formoknál való részletes leírás itt már érthető lesz.

Mezői:

A *Cim* mező text, szerkeszthető, kötelező kitölteni.

A *Eloado*, *Kiadas*, *Megjegyzés* mező text, szerkeszthető.

A *FeltoltoAzon* létrehozáskor számított mező, értéke `@UserName` vagyis az aktuális felhasználó teljes kanonikus neve.

A *Feltoltve* számított mező, értéke `@Date(@Now)` vagyis az aktuális dátum.

A *Pontszam* valamint a *Hozzászólások* mezők number típusúak, szerkeszthetők és csak az üzemeltetők vihetnek be értéket.

A *Fajl* egy Rich Text Lite mező. Elrejtjük, ha nem új dokumentum és szerkesztőmódban van `(!@IsNewDoc & @IsDocBeingEdited)`, hogy később már ne lehessen módosítani a fájlt, csak a jegyzet adatait lehessen szerkeszteni. Talán felvetül a kérdés, hogy miért? Azért, mert ha már fel lett töltve egy jegyzet, akkor ne lehessen kicserélni egy teljesen másikra, főleg ha már írtak is hozzá kommenteket. Ha ki lehetne cserélni, eléggé összekutyulná a dolgokat. Másik kérdés lehet, hogy miért nem az input enablead tulajdonságnál van letiltva ilyenkor az adatbevitel? Azért mert ennek a típusú mezőnek nincs ilyen tulajdonsága.

Eseményei:

A formunknak két eseményét kell megírni. A **Querysave** eseménynél lekérdezni, hogy új-e a dokumentum, a **Postsave** eseménynél pedig ha új, akkor meghívni a szkript könyvtár azon metódusát ami levelet küld annak a felhasználónak aki a Jegyzet *Eloado* mezőjében szerepel, hogy valaki olyan jegyzetet töltött fel ami kapcsolódik hozzá.

Komment form:

4. ábra

Típusa válaszdokumentum. Tehát ezzel a formmal készített dokumentumok majd a jegyzetek válaszdokumentumai, gyerekei lesznek.

Mezői:

A *Nev* mező létrehozáskor számított, alapértelmezett értéke az aktuális felhasználó.

A *Komment* mező text, szerkeszthető, inputja validálva van, kötelező kitölteni.

A *Pontszám* Combo box, szerkeszthető, **kezdőértéke**: `"Nincs"`. Elrejtjük, ha nem új és szerkesztőmódban van, hogy ha szerkesztjük a hozzászólást, akkor a pontszámot már ne lehessen megváltoztatni. Szebbnek tartottam volna, hogy ha az adatbevitel letiltását erre a mezőre, ha nem új a dokumentum. De az input enabled tulajdonságnál, az `@IsNewDoc` valamiért itt mindig igazat adott vissza, ezért választottam az elrejtést.

Az *Ido* pedig számított, alapértelmezett értéke a jelenlegi dátum és idő, `@Now`.

Eseményei:

Queryopen: Az aktuális nézetben kijelölt dokumentum ID-ját eltároljuk, mikor megnyitunk egy komment formmal készült dokumentumot. Ez az ID egy Jegyzet dokumentum azonosítója lesz, amihez hozzászólunk, hiszen az van kijelölve. Erre akkor lesz szükségünk, amikor meg akarjuk állapítani, hogy az adott felhasználó pontozhat-e, mivel azon metódusnak, ami ezt eldönti, a fődokumentum (vagyis a jegyzet, amihez hozzászólunk) azonosítójára van szüksége. Ekkor végighaladva a jegyzet összes gyerekén megnézi, hogy az adott felhasználó pontozott-e már. Ha csak megnyitunk egy hozzászólást olvasásra, vagy

szerkesztésre akkor nyilván ez az ID magának a hozzászólásnak az ID-ja, de ez esetben viszont nem lesz rá szükség úgysem.

Querysave: Ha a dokumentum új és nem pontozhat a felhasználó, mivel már korábban pontozta az adott jegyzetet, akkor bármit is állított be pontértéket, azt „Nincs”-re állítjuk.
Valamint ha a dokumentum nem új, tehát szerkesztette akkor a Szerkesztve mező tartalmát „Szerkesztve”-re állítjuk.

Postsave: Ha a dokumentum új akkor meghívjuk azt a metódust amely értesítést küld a jegyzet tulajdonosának vagyis a *FeltoltoAzon* mezőben szereplőnek, hogy hozzászóltak az általa feltöltött jegyzethez, illetve az *Eloado* mezőben szereplőnek is, hogy hozzászoltak a hozzá kapcsolódo jegyzethez.

Bongeszo form:

Nincs semmilyen esemény megírva. Két text és egy radio button típusú mezőt tartalmaz, amikben a böngésző neve és verziója van tárolva, valamint a radio button mutatja, hogy támogatott-e vagy nem, melyet kötelező megadni.

6.1.2. Nézetek létrehozása

19 nézet található az adatbázisban. Az *Azon_nev* és a *Nev_azon* nézetek szerviz nézetek, amelyek egy név kikeresését teszik lehetővé azonosító alapján, illetve fordítva. Ezek a nézetek nem láthatóak. A *Minden dokumentum* nézetet csak az üzemeltetőknek látszik, esetlegesen a munkájukat segíti. A böngészők nézetek, pedig a böngészőket mutatják. Az alkalmazás szempontjából a lényeges nézetek a *Jegyzetek* és a *Jegyzetek, kategorizált* csoportosított nézetek. Előbbi négy nézetet tartalmaz melyben minden nézet a jegyzet valamely tulajdonsága alapján rendezve van, és tartalmazza a hozzászólásokat is. Utóbbi öt nézetet tartalmaz, melyek kategorizált nézetek a jegyzet valamely tulajdonságai szerint és nem tartalmazzák a hozzászólásokat. Több szót nem is érdemes vesztegetni rájuk, amiket előzményekben írtam a nézetekről abból megérthetőek, létrehozásuk egyszerű. Talán egy dolgot mégis érdemes kiemelni. A válaszdokumentumok megjelenítése és a szelekciós képlet.
Ezt a *Jegyzetek* nézeten magyarázom el. Mivel ebben a nézetben két különböző form-al készült dokumentumokat kell megjeleníteni ezért a szelekciós képletet vagy megadjuk

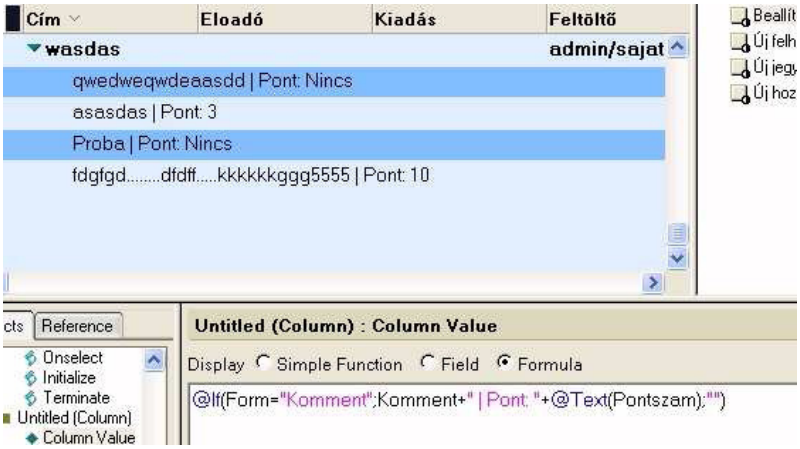
egyszerű művelet segítségével, vagy formulanyelvvel. Én egyszerű művelettel adtam meg, de most ide formula nyelven írom, mert talán szemléletesebb:

```
SELECT ((Form = "Jegyzet") | (Form = "Komment"))
```

Tehát azokat a dokumentumokat jeleníti meg amelyek a *Jegyzet* vagy a *Komment* formmal készültek. A jegyzet tulajdonságait megjelenítő oszlopoknál egyszerűen megadhatjuk a mezőket, amiket meg akarunk jeleníteni. A válaszdokumentumokat megjelenítő oszlop bonyolultabb lesz. Képlettel kell megadni:

```
@If(Form="Komment";Komment+" | Pont: "+@Text(Pontszam);"")
```

Az 5. ábrán látható Cím oszlop előtti feketével kijelölt oszlop, ami a válaszdokumentumok megjelenítéséért felelős.



5. ábra

Amennyiben *Komment* formról van szó, akkor jelenítse meg a *Komment* mező tartalmát, hozzákonkatenálva a „ | Pont:” sztringet, valamint a *Pontszam* mező tartalmát amit szöveggé kell alakítani.

6.1.3. Kód szekció

Osztott műveletek: Itt találhatók a nézetekben és formokon elhelyezett műveletek kódjai.

Szkript könyvtárak: Itt találhatók a LotusScript és JavaScript nyelven megírt kódok, melyek többek között felelősek a levélküldésekért.

Adadbázis szkriptek: Itt az adatbázis események találhatók. A következő események vannak megírva: Querydocumentdelete, Querydocumentundelete, Postdocumentdelete.

- **Querydocumentdelete**: Úgy van megírva, hogy ha az adott felhasználó, aki törölni akar nem üzemeltető, az esetben dokumentum csak akkor legyen törölhető, ha ez

jegyzet. Nem kell törődni azzal, hogy ez a felhasználó tulajdonosa-e a dokumentumnak. Ugyanis emlékezzünk vissza, a *Jegyzet* formon a *FeltoltoAzon*, vagyis a feltöltő azonosítója, aki a dokumentumot létrehozta, egy author típusú mező. Így csak a saját jegyzetét van joga szerkeszteni a felhasználónak, és így egyúttal törölni is csak a saját jegyzetét tudja. Az *uzemelteto* szerepkörrel rendelkezők pedig bármit törölhetnek. Azonban ha egy üzemeltető felhasználót akar törölni, akkor az egy kicsit megbonyolítja a dolgunkat, mivel úgy lenne a korrekt, ha az ACL-ből is törlődne, valamint a profil dokumentuma is. Ennél az eseménynél nem írható meg a törlés, mert ez az esemény csak arra a kérdésre ad választ, hogy meg lehet-e jelölni egy dokumentumot törlésre vagy sem. Ha itt írnám meg a törlést is, akkor már a kijelöléstől törlődne az adott felhasználó vagy felhasználók az ACL-ből. Ami nem lenne jó, ha mégis visszavonnám a kijelölést. Ezért a törlést majd a Postdocumentdelete eseménynél kell megírni. Ahhoz viszont, hogy ott tudjuk, hogy mit kell kitörölni ezért, el kell tárolni ezeket az információkat. A törlésre kijelölt felhasználók azonosítóját, ezért beteszem egy a Javaban Vector néven ismert adatszerkezetbe, amihez majd a Postdocumentdelte eseménynél hozzáférek. (Vector nem létezik a LotusScript-ben, ezért azt külön meg kell írni, és deklarálni a *Declarations* résznél, hogy minden eseményből látható legyen, és az adatbázis inicializálásnál példányosítom, így csak egy példány jön létre belőlle a megnyitott adatbázispéldányhoz.)

- **Querydocumentundelete**: Amennyiben megszüntetjük a törlésre való kijelölést, akkor fut le ez az esemény. Ez esetben ki kell venni a Vectorból a felhasználót, aki már nincs kijelölve.

- **Postdocumentdelete**: Akkor fut le miután törlődik egy dokumentum az adatbázisból. Itt kell megírni, hogy akiknek a felhasználó azonosítója benne van a Vectorban, azokat töröljük az ACL-ből, valamint töröljük a profildokumentumát is.

Ügynökök: Öt darab ügynök található: A zárójelben lévők csak akkor indulnak el, ha kézzel elindítjuk őket, vagy egy kódban valahol meghívjuk. Ezek a következők: *(Torol_user_profil)*: Törli az összes felhasználó profilját. Erre inkább csak a fejlesztésnél volt szükség.

(levelKommentek), (levelUjdok): Csak annyit tesznek, hogy meghívják a levélküldésért felelős metódusokat, amik a szkript könyvtárban foglalnak helyet. A webes megjelenés miatt szükségesek, mivel ott csak JavaScript használható és JavaScriptből nem lehet LotusScriptet hívni közvetlenül, viszont LotusScriptben írt ügynököt igen.

A *Pontszamol* ügynököt új dokumentum keletkezésének hatására indul el. Ha egy jegyzethez új hozzászólás érkezett újraszámolja a pontértékét a jegyzetnek, és módosítja amennyiben szükséges.

A *Torol_arvak* egy ütemezett ügynök. Az árva dokumentumokat törli, amelyeknek nincsenek szülőik. Ilyenek lehetnek azok a hozzászólások, amelyek olyan jegyzethez tartoznak, amelyek törölve lettek. Persze megoldható lett volna, hogy a jegyzet törlésével törlődjenek vele együtt a hozzá tartozó hozzászólások is, ám szerettem volna, hogy egy ütemezett ügynök is legyen az adatbázisban. Így ez arra a logikára épül, hogy ha törlünk egy jegyzetet, nem kell rögtön törölgetni utána a gyermekdokumentumait, hanem csak időközönként mikor összegyűlt már valamennyi szemét. Valamint egyszerűbb is volt így megoldani, különben hasonlóan kellett volna mint a felhasználó törlése esetén. De ott szükséges volt, hogy ha töröljük, akkor a hozzáférése is szűnjön meg rögtön, itt nem sürgős rögtön törölni a válaszokat.

6.2. A webes felület

XPage-ekből épül fel, tíz darabból. Nem beszélhetünk nézetekről és formokról, weboldalakról lesz szó, melyek nézeteket vagy a formok által létrehozott dokumentumokat jelenítik majd meg. Minden XPage egy weboldal lesz.

Xp_navigacio:

Ezen az oldalon kaptak helyet az alábbi képen is látható vezérlő elemek (linkek, gombok):

6. ábra

Akármelyik elemre kattintunk egy másik oldalra jutunk. A navigáció azért lett külön oldalként elkészítve, mivel ez minden oldal felső részén szerepelni fog majd, így egyszerűen include page-ként beszúrható rájuk. Az include page olyan mint a formoknál a subform.
Hogy egy link megnyisson egy másik oldalt arra több lehetőségünk is van: Megadhatjuk a megnyitni kívánt oldal URL-jét, megadhatjuk legördülő listából az oldal nevét, vagy akár link OnClick eseményéhez rendelhetünk egy műveletet ami megnyit egy oldalt, amit megtehetünk akár egyszerű művelettel, vagy JavaScripttel, vagy a kettőt vegyítve. Mit is értek vegyítés alatt? A Notes kliensbeli felületen már láthattuk, hogy bizonyos műveleteket nem kell kódolnunk, csak egyszerűen megadhatjuk úgy, hogy kiválasztjuk valamilyen legördülő listából, esetleg paraméter is megadható egy szövegdobozban. Itt is megvan ez a lehetőség, csak bővül annyival, hogy nem csak konstans értékeket lehet megadni paraméterként. Szinte minden helyen, ahol értéket lehet megadni, lehet számított is, vagyis megírhatunk egy JavaScript kódot, ami kiszámítja az értéket. Így megtehető, hogy a műveletet kiválasztjuk a legördülő listából, például, hogy nyisson meg egy oldalt, de a paramétert, hogy melyik oldalt azt egy JavaScript kóddal számoltatjuk ki, ha nem tudjuk előre, hogy melyik oldalt akarjuk megnyitni, mert futás közben dől el.
Az egyes vezérlőknek szabályozhatjuk még a láthatóságát, módosíthatóságát, és a feldolgozásban való részvételét. Ezek a visible, read-only, és disabled checkboxok. Ki-be pipálhatjuk őket, illetve írhatunk hozzájuk kódot is ha dinamikusan akarjuk, hogy ez változzon.

Megjegyzés: Mivel ez még egy nagyon új technológia, olyannyira, hogy még dokumentáció is elég hiányos róla, így hibák is vannak benne. Egyik hiba, pont ezen a részen van. Mivel, ha azt mondjuk egy vezérlőnek, hogy ne legyen látható vagy írásvédett, legyen, ebbe ő beleérti azt a tulajdonságot is, hogy nem vesz részt a feldolgozásban, hiába azt nem állítottuk be. Nincs mit magyarázni, egyértelműen látható, hogy ennek így nem sok értelme van, tehát ez egy hiba. Sajnos nem ez az egyetlen hiba, több helyen is említenek hibákat az Interneten. De hát még nincs kiforrva ez a technika, az újabb verziókban valószínűleg javítva lesz, illetve bizonyos dolgok már javítva vannak, de én ehhez a verzióhoz jutottam hozzá.

Vezérlői:

A *„bejelentkezve” utáni link* címkéje dinamikusan változik és az aktuális felhasználó azonosítóját mutatja, JavaSrcipttel számoltatott. A link típusa URL, ami szintén számoltatva van, az aktuális felhasználó adatlapját nyitja meg. Anonymous felhasználó esetén a kezdőoldalt.

A *Bejelentkezés* link egy URL parancsot fog tartalmazni: `/Xp_kezdo.xsp?OpenDatabase&login`.
Valamint elrejtjük, ha a felhasználó be van jelentkezve

A *Kijelentkezés* linket elrejtjük, ha a felhasználó nincs bejelentkezve, vagyis Anonymous. A kijelentkezés URL parancs nem működik megfelelően, valamiért nem zárja le az adott sessiont, így egy vissza gomb nyomásával simán vissza lehet térni bejelentkezés nélkül. Ezért az **OnClick** eseményénél egy figyelmeztető üzenet kiíratását helyeztem el, hogy zárja be a böngészőt. A böngésző bezárása, majd újbóli megnyitás után már nem lehet bejelentkezés nélkül visszatérni egy olyan oldalra ami bejelentkezést igényel, akkor sem ha nem törölte az előzményeket a felhasználó.

Megjegyzés: A Notes levelezésének webes felületén ugyanez a helyzet, amely beépített és nem általam készített, ezért valószínű, hogy nem én csináltam valamit rosszul.

A *Kezdőlap, Felhasználók, Jegyzetek* valamint a *Keresés* linkeknél egyszerűen csak kiválasztjuk, hogy nyissa meg az ezeknek megfelelő oldalt, az Open Page legördülő listánál.
A *Levelezés* linket elrejtjük, ha a felhasználó Anonymous, az URL-ét pedig számoltatjuk.

Az *Új jegyzet* és a *Beállítások* gombot elrejtjük, ha a felhasználó Anonymous. Az *Új jegyzet* **OnClick** eseményénél megadjuk egy egyszerű művelet segítségével, hogy nyissa meg az *Xp_jegyzet* oldalt szerkesztőmódban. A *Beállítások* gomb **OnClick** eseményénél pedig szintén egy egyszerű művelettel adjuk meg, hogy nyissa meg a felhasználó beállításait megjelenítő oldalt.

Az **Xp_kezdo** és az **Xp_letoltes** nem igényel különlegesebb magyarázni valót. A kezdő oldalon semmi más nem található a navigációs oldalon kívül, minthogy kiírja, hogy támogatott-e a böngészőnk vagy nem. De akár tovább bővíthető még tetszőleges információk megjelenítésével. A letöltés oldalon pedig csak egy letöltés vezérlő jelenik meg, hogy le tudjuk tölteni a fájlt.

Xp_felhasznalo:

A felhasználó adatlapját jeleníti meg és ad lehetőséget a szerkesztésre. Itt már szükséges adatforrást definiálni. Adatforrást több helyen is definiálhatunk. Definiálhatjuk az oldal Data tulajdonságánál, vagy a Data ablakban (Window->Show Eclipse views->Data).

7. ábra

Az adatforrás definiálásakor meg kell adni a formot, amellyel készült az adott dokumentum, valamint kiválasztható a default action: create,edit,open. Valamint megadható a dokumentum ID-ja. Ez utóbbi nem kötelező, mert nem tudhatjuk, hogy melyiket fogjuk megnyitni. Ezt megnyitáskor is megkaphatja. Ahhoz, hogy megjelenítsük és szerkeszteni is tudjuk a dokumentumot, szükség van olyan vezérlőkre melyek értékek megjelenítésére és szerkesztésére használhatók, mint például: Edit box, Combo box, stb.

Ha a fentebb leírtak közül a második módon definiáltuk az adatforrást, akkor Data ablakban (8. ábra) már meg is jelennek az adatforrás mezői, amiket csak rá kell húzni az oldalra amire, mint Edit box (a 7. ábrán látható egyszerű szövegdobozok) kerül rá összerendelve a megfelelő mezővel. Amennyiben nem Edit boxként akarjuk rátenni az oldalra, úgy kattintsuk rá a 8. ábrán pirossal bekarikázott ikonra. Ezután ha ráhúzzuk az oldalra a mezőt, akkor felugrik egy ablak, ahol választhatunk milyen vezérlőként kerüljön rá az oldalra.

8. ábra

Ha nem így tettünk, akkor külön is rápakolhatjuk a megfelelő vezérlőt, és megadhatjuk a Data tulajdonságánál, hogy melyik mezővel legyen összerendelve

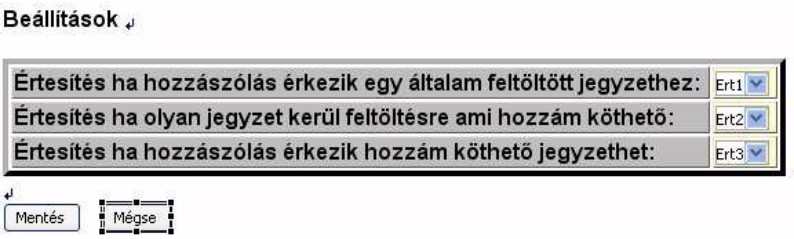
Ha egyik fenti módon se definiáltuk az adatforrást, azt a hozzárendelésnél, a vezérlő Data tulajdonságánál szintén megtehetjük. Tehát akár három helyen is van rá lehetőségünk. Ha az oldalon bárhol már definiáltuk az adatforrást, legyen az akár egy vezérlő Data tulajdonságánál, a hozzárendelésnél, a többi vezérlőnél már nem kell újra definiálni csak kiválasztani, az adatforrás nevét és, hogy melyik mezőhöz rendeljük hozzá.

Vezérlői:

A *Nev, Azonosito, E-mail, Honlap, Telefon, Tipus* Edit boxok. Az *Azonosito* valamint a *Tipus* read-only. A *Tipus* itt a Notesos változattól eltérően csak Edit box, mivel a típust úgy sincs jogunk megváltoztatni, a megjelenítéshez pedig ez is elegendő. Akár egy Computed fielddel is megjeleníthettük volna. A *Honlap* Edit box mellett van még egy link is, azért mivel az Edit box segítségével meg tudjuk jeleníteni és szerkeszteni a honlapot, de rákattintással nem tudjuk megnyitni, ehhez már link kell. A *Honlap* Edit box tehát akkor látszik, ha szerkesztő módban van, csak szerkesztéshez használjuk, a megjelenítéshez a linket használjuk. A *LegIskVeg* Combo box, melynél meg kell adni a Values tulajdonságánál, a választható értékeket akár csak a Notesos párjánál. A három Checkbox read-only lesz. Ezeknél a Data tulajdonságnál nem csak azt kell megadni, hogy melyik mezővel legyenek összerendelve, hanem azt is, hogy mi legyen az értékük, ha be vannak pipálva, illetve ha nem. Vagyis ha be vannak pipálva, akkor értékeik rendre: „Üzemeltető", „Létrehozhat dokumentumot", „Törölhet dokumentumot". A nincsenek bepipálva mezőt pedig üresen hagyjuk.

Ahhoz, hogy a módosítások átkerüljenek az adatbázisba, el kell küldenünk neki. Ugyanis mikor egy ilyen oldalon operálunk az adatokkal, az olyan, mint amikor Notes kliensben egy front-end dokumentumon operálnánk. Ott az elmentéssel kerülnek át az adatok a back-end dokumentumra. Itt is szükséges tehát elküldeni az adatot a Notesnak, a szervernek. Ehhez csak el kell helyezni egy gombot az oldalon és Submit típusúra állítani. Ez lesz a *Mentés* nevű gomb. El kell még helyezni egy *Szerkeszt* gombot is az oldalon amivel szerkesztő módba tudjuk rakni. Ez egy normál gomb lesz aminek az **OnClick** eseményéhez megadunk egy olyan művelet, hogy tegye át szerkesztőmódba. Ez az művelet szintén megadható egyszerű művelet segítségével. Valamint egy *Mégse* gombot is elhelyezünk, ami kilép a szerkesztőmódból. Ez utóbbi nem kötelező, mivel ha egy másik oldalra lépünk akkor úgy is kilépne szerkesztőmódból mentés nélkül.

Xp_Beallitasok:

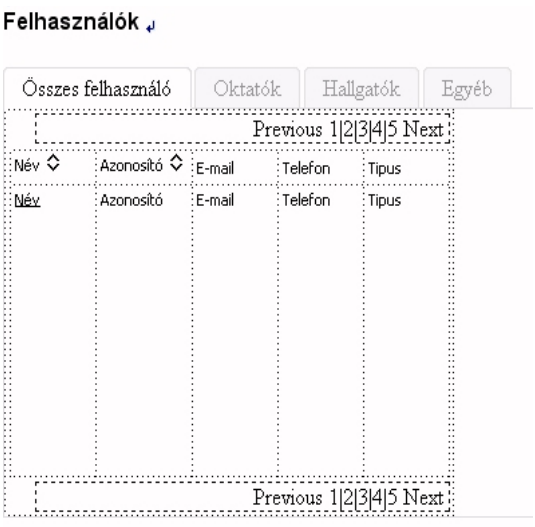


Ezen az oldalon nem igazán van részletezni való, úgyhogy ezt nem is tenném meg.

9. ábra

Definiáljuk az adat forrást, amely a Beallitasok_profil form lesz, elhelyezünk három Combo boxot amivel, a három beállításunkat tudjuk módosítani. Itt nem szükséges a *Beallitasok_profil* form author mezőjéhez semmilyen vezérlőt társítani mivel azt úgysem módosíthatjuk és a megjelenítése is szükségtelen. Valamint elhelyezzük a *Mentés* és a *Mégse* gombokat. Mindkét gomb **OnClick** eseményéhez megadjuk, hogy nyissa meg a kezdő oldalt. Ezek a gombok mindig láthatóak lesznek, mert a dokumentumot mindig szerkesztő módban nyitjuk meg, mivel beállításainkat úgyis csak mi láthatjuk, így nem kell külön még szerkesztőmódba is rakni, ha meg akarjuk változtatni. Ezt a fentebb említett default actionnél tudjuk megadni, amikor a forrás hozzárendelését végezzük.

Xp_felhasznalok:



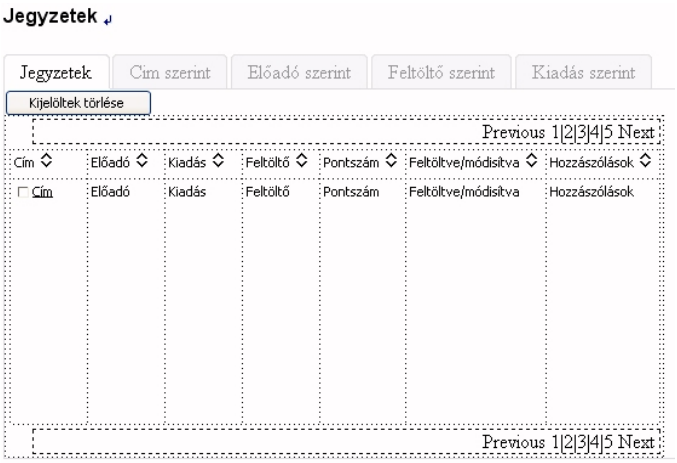
Szintén nem igényel részletezni valót, mint ahogy az ezt követő nézeteket tartalmazó oldal sem. Egy füles panel található rajta annyi füllel, ahány felhasználókkal kapcsolatos nézetünk van a Notesban. Minden fülbe egy nézet kerül. Nem kell adatforrást definiálnunk az oldalhoz, ezt majd az adott nézet létrehozásakor megadjuk, melyik Notes nézetből vegye az adatokat. A füles panel semmi különösebb tulajdonsággal nem rendelkezik, ellenben a nézettel.

10. ábra

A nézet oszlopcímre kattintva beállíthatjuk, hogy az adott oszlop rendezhető legyen. Itt fontos figyelni arra, hogy ha a Notes nézetben, amiből veszi az adatokat nincs beállítva, hogy

rendezhető legyen az oszlop, akkor itt hiába állítjuk be, az nem lesz rendezhető. Valamint a rendezés iránya is a Notes nézet beállításaitól függ. Ahhoz, hogy ne csak mutassa a nézet a dokumentumokat, hanem belőle meg is tudjuk nyitni őket, szükséges, hogy legalább egy oszlopában beállítsuk a következőt: „Show values in the column as link”. Ez alatt még beállíthatjuk azt is, hogy olvasható vagy szerkesztő módban nyissa meg. Ám ez még nem elég, ha ilyenkor a linkre kattintanánk hibaüzenetet kapnánk, mivel meg kell még neki adni, hogy melyik oldalt nyissa meg ilyenkor. Ezt a nézet első tulajdonságánál (View) adhatjuk meg: „At runtime selected document using:” legördülő listából kiválasztva az *Xp_felhasznalo* oldalt. Ha ezeket beállítottuk, akkor a nézetben az adott oszlopban az értékek linkként jelennek meg, amire rákattintva megnyílik az *Xp_felhasznalo* oldal az adott dokumentum adatival. Ezeket beállítjuk mindegyik, az oldalon elhelyezett nézetre. Mint korábban említettem az adatforrás definiálásánál, meg lehetett adni a dokumentum ID-t, hogy melyiket nyissa meg, de írtam, hogy erre nincs szükség, mert majd azt megnyitáskor megkapja. Így egy nézetből megnyitva nem is kell törődnünk ezzel, ő tudja, hogy melyiket kell megnyitni. Egyes speciálisabb esetekben kell csak nekünk kézzel megadni a megnyitandó dokumentum ID-ját.

Xp_jegyzetek:



11. ábra

Az *Xp_felhasznalok* analógiájára hasonlóan hozzuk létre ezt az oldalt. A Notes jegyzet nézetek közül a kategorizáltakat fogjuk itt megjeleníteni. Elhelyezünk egy füles panelt annyi füllel ahány kategorizált nézetünk van, majd ezek mindegyikébe egy nézetet a megfelelő Notes nézethez hozzárendelve.

Kategorizáltságra vonatkozóan itt semmit nem kell beállítani. Ha a Notes nézet kategorizált, akkor itt is kategorizáltként fog megjelenni. Hasonlóan beállítjuk, hogy megnyithatók legyen belőlük a dokumentumok. Természetesen ezt nem az első oszlopra tesszük meg mert az a kategória, hanem azt követőre, vagy ha tetszik azt követőkre. Annyi többletet csinálunk, hogy beállítjuk még a kategória oszlopot követő oszlopra, hogy jelenjen meg egy Checkbox. Ezt az

adott oszlop tulajdonságainál állíthatjuk be. Valamint elhelyezünk minden fülben még egy gombot, *Kijelöltek törlése* címkével. Ennek a gombnak az **OnClick** eseményéhez hozzáadunk egy egyszerű műveletet, mégpedig „Delete selected documents” műveletet. Ennek egy kötelező argumentuma van: a nézet, amiből törölni akarunk. Valamint egy opcionális, egy megerősítő szöveget adhatunk meg. Ezután képesek leszünk a dokumentumokat a Checkboxszal kijelölni és a gomb megnyomásával törölni.

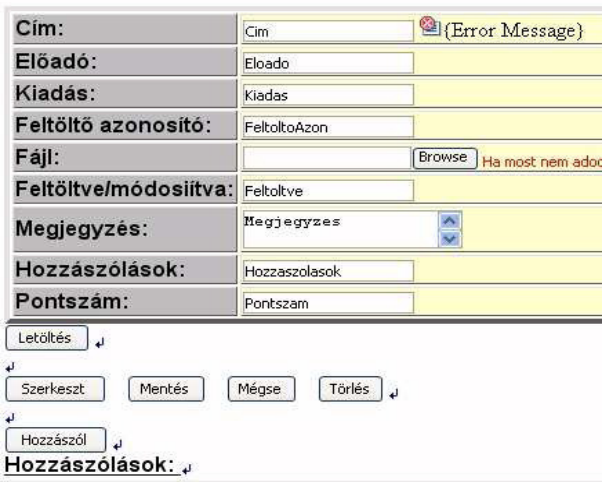
Megjegyzés: Bár a kategorizáltságra nem kell semmi külön beállítást tenni, ám amennyiben válaszdokumentumokat is meg akarunk jeleníteni akkor igen. Annak az oszlopnak a tulajdonságainál, ami a válaszdokumentumok megjelenítéséért felelős be kell kattintani az „Indent responses” checkboxot.

Xp_jegyzet:

Ez a legösszetettebb oldal. Itt jelennek meg a jegyzet adatai, illetve a jegyzetre írt hozzászólások is.

Lássuk elsőnek a jegyzet adatainak megjelenítését és vezérlőit:

Definiáljuk az adatforrást, a jegyzet formot. Helyezzük el a megjelenítéshez és szerkesztéshez szükséges Edit boxokat. Az *Eloado* és a *Kiadas* Edit boxok, a *Megjegyzes* pedig Multiline edit box. A *FeltoltoAzon*, a *Feltoltve,* a *Hozzaszolasok*, és a *Pontszam* read-only legyen, valamint a *Falj*-nak ne Edit boxot , hanem egy File uploadert tegyünk, hogy fel lehessen tölteni a fájlt és rejtsük el ha a dokumentum szerkeszthető és nem új, hogy később ne lehessen több fájlt csatolni.



12. ábra

A *Cim* Edit box tartalmát validálni kell, hogy biztosítsuk, hogy mindenképp ki legyen töltve. Ehhez válasszuk a *Cim* Edit boxot, majd kattintsunk a Validation tulajdonságára. Pipáljuk be, hogy „Required field” és írjuk be a hibaüzenetünket az alatta elhelyezkedő szövegdobozba. Ha nem írunk a szövegdobozva semmit, a „Field required” üzenet fog kiíródni.

Ez jelenti azt, hogy a mezőnek kötelező kitöltve lennie. Ugyanitt a Validation tulajdonságnál hossz szerinti validációt is megadhatunk. Ekkor ha a címet nem töltjük ki és el akarjuk küldeni az oldalt a szervernek, a böngésző fel dob egy ablakot a hibaüzenetünkkel. Ez a kliens oldali validáció. Ám ezt lehet tovább is finomítani. Az alkalmazás beállításaiban az XPages fülön kapcsoljuk ki a kliens oldali validációt, és a vezérlők közül válasszuk az Error Message-et és helyezzük el a *Cim* Edit box mellett (Máshol is elhelyezhetnénk). Állítsuk be, hogy a *Cím*-hez mutassa a hibaüzenet. Ezt a vezérlő első fülét választva, a „show error messages for:" legördülő listánál tehetjük meg. Ekkor ha nem töltjük ki a címet nem dob fel a böngésző külön ablakot, hanem ott ahol elhelyeztük ezt a vezérlőt az oldalon, ott jelenik meg a hibaüzenetünk.

Megjegyzés: Lehetőségünk van több fajta validátor közül is választani, sőt saját validátort is létrehozni. Ezt a validálni kívánt eszköz All properties fülén a validatorsnál tehetjük meg. Amint korábban is említettem nem egy hiba található még ebben a technikában, itt is van hiba. A validateExpression nem müködik megfelelően, pedig ezzel lehet ellenőrizni például, hogy egy adott részlet előfordul egy Edit box-ban megadott szövegben vagy nem. A File uplodernél egyáltalán nem működik a validáció. Amennyiben beállítjuk, hogy „Required field", ő mindig üresnek fogja venni, hiába van kitöltve. Ezért oda csak én írtam ki egy statikus figyelmeztető szöveget. Valamint a Multiline edit box esetén hiába adjuk meg a hibaüzenetünket, akkor is csak azt írja ki, hogy „Field required".

Helyezzük el a következő gombokat. *Letöltés, Szerkeszt, Mentés, Mégse, Törlés, Hozzászól*.
Letöltés: Az **OnClick** eseményénél megadunk egy egyszerű műveletet, hogy nyissa meg a letöltés oldalt. Itt már szükséges lesz megadni a dokumentum ID-t, amit JavaScriptel könnyedén lekérdezhetünk.

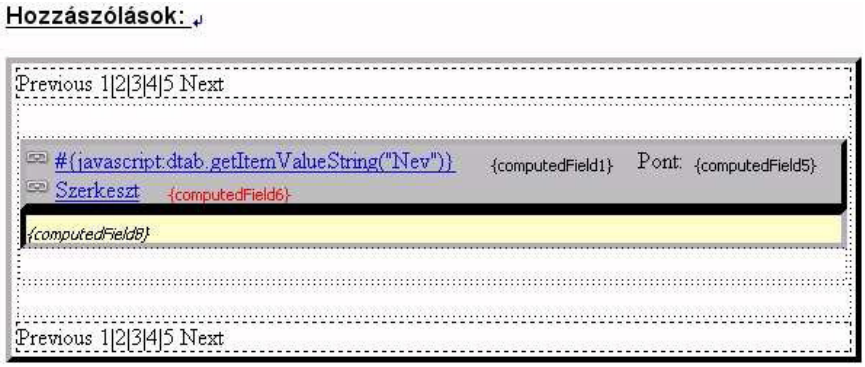
Szerkeszt: **OnClick** eseménynél egy egyszerű műveletet adunk meg, váltson szerkesztőmódba. Valamint elrejtjük, ha az aktuális felhasználó nem a dokumentum tulajdonosa, feltöltője.

Mentés, Mégse: Ezek egy Submit és egy Cancel típusú gombok, **OnClick** eseménynél pedig megnyitjuk az *Xp_jegyzetek* oldalt. Ha szerkesztő módban van a dokumentum elrejtjük őket.

Törlés: **OnClick** eseménynél egy „Delete Document” művelet. Két argumentuma van. Az első a következőnek megnyitandó oldal: *Xp_jegyzetek*, valamint egy megerősítő szöveg. Elrjetjük, ha az aktuális felhasználó nem a dokumentum tulajdonosa.

Hozzászól: **OnClick** eseménynél szintén egy egyszerű művelet: „Create response document”. Két argumentuma van. A megnyitandó oldal ami az *Xp_komment* lesz, és a szülő dokumentum ID-ja amit szintén JavaScripttel lekérdezünk.

Most nézzük a hozzászólások megjelenítését és vezérlőit:



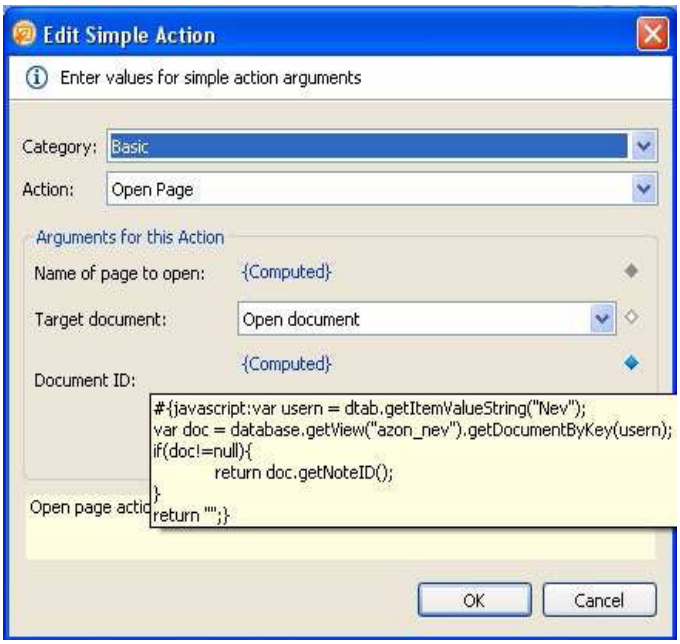
Egy Adat táblával (Data Table) segítségével fogjuk megjeleníteni. Ez is egy konténervezérlő, a nézethez hasonlítható.

13. ábra

Használata nehezebb és komplikáltabb, mint a nézeté, de sokkal bonyolultabb megjelenítést tesz lehetővé. Nem kell sorokból és oszlopokból állnia, mint a nézetnek, mi állíthatjuk össze a megjelenését. Tehát adjunk az oldalhoz egy adat táblát. Szüksége van adatforrásra, egy kollekcióra. Ezt JavaScriptel adjuk meg. Lekérjük az adott jegyzet válaszdokumentumainak kollekcióját, vagyis a kommentjeit. Az adattábla egyesével végig fog menni a kollekció minden elemén, amiket mi így meg tudunk jeleníteni. Ugyanitt kötelező még megadni a következőket. Collection name, a kollekció neve. Ezzel hivatkozunk majd a kollekcióra. Valamint a Colleciton index. Ezzel az indexeire. Itt megadható még a kezdő index, hogy a kollekció hányadik elemétől kezdje megjeleníteni a kollekciót (alapértelmezésben a legelsőtől kezdi), illetve egy ismétlési érték, ami azt jelenti, hogy egy oldalon hányat jelenítsen meg.
Ekkor még teljesen üres a tábla, szúrjunk be egy oszlopot. Ebben az oszlopban helyezzünk el három panelt egymás alatt. A szürkét, a sárgát és az utolsó átlátszót. Ezek lesznek egy hozzászólás fejléce, tartalma, és egy sorköz a hozzászólások között. Ezután így fogok rájuk hivatkozni itt a szövegben.

A fejlécen fogjuk megjeleníteni a hozzászóló azonosítóját linkként, amelyre rákattintva megnyílik a felhasználó adatlapja; a hozzászólás idejét; a pontszámot; azt, hogy szerkesztve volt-e; valamint egy linket a szerkesztésre.

A felhasználó azonosítójának megjelenítése: Ez egy link lesz, címkéje az aktuális kollekcióelem *Nev* mezője. Az aktuális kollekcióelemre a nevével hivatkozunk. Olyan, mint egy mutató, mindig az aktuális kollekció elemet címzi, és saját magát lépteti. A link **OnClick** eseményéhez egy oldal megnyitásnak műveletét rendeljük. Ez az oldal az *Xp_felhasznalo* lesz amennyiben nem Anonymous (nem regisztrált) felhasználóról van szó. Anonymous esetén a kezdő oldal, ezt JavaScripttel adjuk meg. A megnyitandó dokumentum ID-ját pedig szintén JavaScripttel tudjuk meghatározni. Itt látható a vegyítés. Egy egyszerű műveletet adunk meg, az oldal megnyitását, de a paramétereit már JavaScripttel határozzuk meg. (14. ábra)



A hozzászólás idejének, a pontszám, szerkesztve volt-e megjelenítése: Mindegyik egy számított mező lesz (Computed field), adatforrása az aktuális kollekció elem *Ido*, *Pontszam*, és *Szerkesztve* mezője.

Link a szerkesztéshez: Szintén egy egyszerű művelet az **OnClick** eseménynél, az *Xp_komment* oldal megnyitása szerkesztőmódban, a dokumentum ID-ja pedig számoltatva van.

14. ábra

Akkor látszik, ha az akuális felhasználó nem Anonymous és övé a hozzászólás (felhasználóneve megegyezik az aktuális kollekcióelem *Nev* mezőjével).

A tartalom részben jelenítjük meg a hozzászólás szövegét. Egy számított mezőt fog tartalmazni. Adatforrása az aktuális kollekció elem *Komment* mezője.

Ezzel teljes az adattáblánk. De még nem teljes az oldalunk. Ugyanis ha új jegyzetet hoz létre valaki levelet kell küldenünk. Ezt a Notesban a form Postsave eseményénél valósítottuk meg.

Mint ahogy a formnak az XPage-nek is vannak ilyen eseményei, ezért itt is az ezen eseményeket használjuk fel erre.

Eseményei:

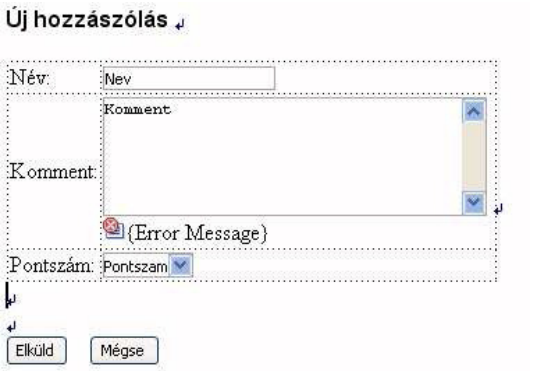
Tehát itt is ezeknél az eseményeknél írjuk meg, pontosabban írnánk, mert egyszer már megírtuk ezeket a metódusokat, csak megkellene hívni őket. Ezt már nem tudjuk egyszerű műveletekkel megvalósítani. A teljes műveletet magunknak kell megírni. Viszont van egy kis probléma: JavaScriptből nem lehet LotusScript metódust hívni. De emlékezzünk, van ügynökünk, ami nem tesz mást, csak meghívja az adott metódust. Direkt azért lett létrehozva, mivel JavaScriptből el lehet indítani bármilyen nyelven megírt ügynököt. Így csak a megfelelő ügynököt kell meghívunk, ha a dokumentum új. Azt, hogy új-e a dokumentum, mint ahogy Notes kliensben is, a **Querysave** eseménynél kérdezzük le. Ám itt nincs *Declarations* rész, hogy globális változót deklarálhatnánk amely minden eseményből látszik. Itt ezt hatásköri változó segítségével tudjuk megtenni.(scoped variable). Egy ilyen változónak, ha egy helyen értéket adunk egy másik helyen kiolvashatjuk. Négy fajtája van attól függően, hogy meddig él: *Application scope, Session scope, View scope, Request scope*. Azt hiszem ezek nem szorulnak magyarázatra. A **Postsave** eseménynél pedig meghívjuk a levélküldésért felelős ügynököt ha a dokumentum új.

Van azonban még egy fontos dolog amit be kell állítani. A *FeltoltoAzon* és a *Feltoltve* read-onlyra vannak állítva, hogy a felhasználó ne tudja megváltoztatni, és mint már említettem, ami read-only az nem vesz részt a feldolgozásban. Hogy fog akkor ezen vezérlők értéke átkerülni az adatbázisba? Ez egyszerűen megoldható. Be lehet állítani, hogy miután elmentünk egy xsp dokumentumot, tehát mikor elküldi az oldal tartalmát a szervernek és az elmenti akkor, számolja újra a notesos (back-end) dokumentumot. Az újraszámolás azt jelenti, hogy végrehatja a default value, translation és validation formulákat. A *Jegyzet* formon pedig *FeltoltoAzon* alapértelmezett értéke a `@UserName`, vagyis az aktuális felhasználó, a *Feltoltve* alapértelmezett értéke pedig `@Date(@Now)`, vagyis a mostani dátum. Az újraszámolást a következőképp lehet beállítani:

All Properties->data->data->xsp dokumentumunk neve->ComputeWithForm tulajdonságát kell „onsave”-re állítani.

Ezzel kész is a jegyzet oldalunk.

Xp_komment:



Ezek után ezen oldal elkészítése már könnyű lesz. Ez eddig megismert módokon elhelyezzük a vezérlőket, gombokat, beállítjuk a láthatóságokat.

15. ábra

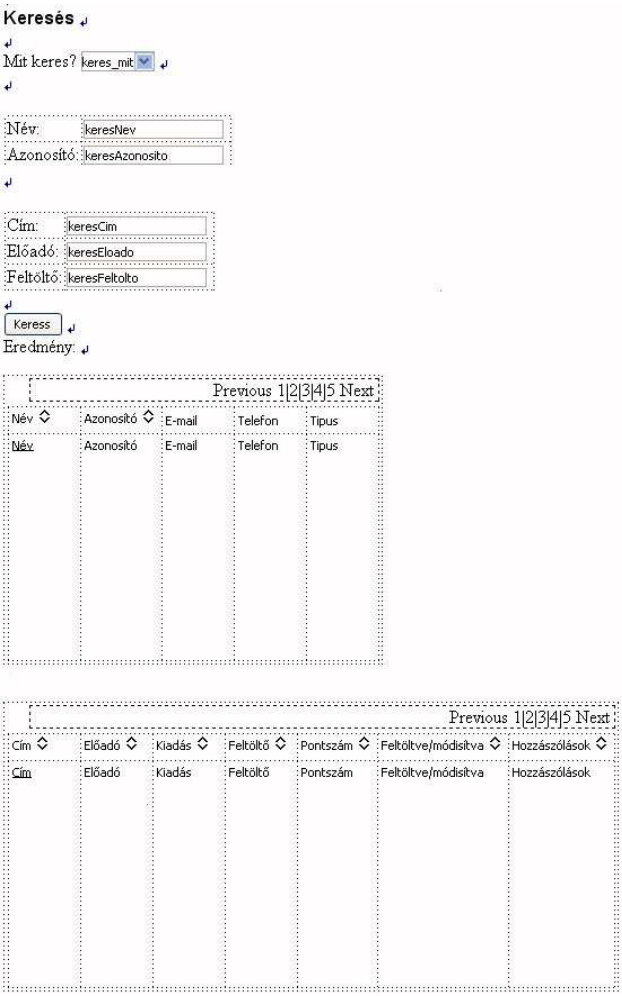
Vezérlői:

A *Nev* read-only, emiatt a ComputeWithForm tulajdonságot szintén „onsave”-re kell állítani. A *Komment* validálva van, kötelező kitölteni, valamint a *Pontszam* el van rejtve, ha nem új a dokumentum. A gombok mindig láthatók, mivel mindig szerkesztőmódban lesz a dokumentum, mert ezt az oldalt hozzászólás létrehozására illetve szerkesztésére használjuk, a megjelenítésre nem. A két gomb Submit illetve Cancel típusú és **OnClick** eseményénél megnyitjuk azt a jegyzetet, amelyik meg volt nyitva mielőtt a hozzászól gombra, vagy a hozzászólásnál a szerkesztés gombra kattintottunk volna.

Eseményei:

Ahogy az előző esetben itt is ügynököt kell hívnunk. Kivétel annak a megállapítása, hogy pontozhat-e a felhasználó **querySaveDocument** eseménynél. Van egy ilyen Lotusos függvényünk, ami ezt megállapítja, csakhogy ezt nem hívhatjuk meg egy ügynökből, mert hiába is adná vissza az értéket a hívás helyére az ügynöknek, az ügynök már nem képes értéket visszaközvetíteni. Ezért ezt a *pontozhat* függvényt meg kell írni külön JavaScriptben is. Ez a Javas szkript könyvtárban kap helyet. Ahhoz, hogy meg tudjuk hívni szükséges, hogy kiadjuk `import szkriptkönyvtárnév` utasítást a szkript szerkesztőben, vagy az oldal Resources fülén, hozzáadjuk ezt a szkript könyvtárat az oldalhoz. A levélküldést, ha új a dokumentum, a **postSaveDocument** eseménynél viszont szintén elintézhetjük egy ügynök hívásával. Ha pedig nem új akkor módosítjuk a Notesos dokumentum (a back-end dokumentum) *Szerkesztve* mezőjét „Szerkesztve”-re.

Xp_kereses:



16. ábra

Ezen az oldalon példát láthatunk arra, hogy lehet keresni és az eredményt egy nézetben megjeleníteni. Az oldalon található egy Combo box, amiben kiválaszthatja az ember, hogy mit keres, felhasználót vagy jegyzetet. Ettől függően jelenik majd meg az oldalon található két nézet közül a felhasználókat vagy jegyzeteket tartalmazó. Ezt egyszerű elérni. A Combo boxban kiválasztott értéket egy hatásköri változóban tároljuk, majd ennek értékétől függően változtatjuk a nézetek láthatóságát. A keresési feltételeket Edit boxokba írhatjuk be melyek szintén hatásköri változókhoz vannak rendelve. Ahhoz, hogy a keresés megtörténjen, a nézet All properties->data->data->search tulajdonságát kell megfelelően megadni.

Nincs más dolgunk, mint a hatásköri változók értékeit kiolvasni és ezekből összeállítani egy kérést (query string), és ezt visszaadni. Ez alapján ő kiválasztja a dokumentumokat, amelyek eleget tesznek a kérésnek, és csak azokat jeleníti meg a nézetben. Tehát nem kell nekünk egy komplett kereső algoritmust írni, csak a kérést összeállítani. A *Keress* gomb pedig egy egyszerű Submit típusú gomb. Erre a Notes súgójában található egy konkrét példakód is.

6.3. A konkurens hozzáférés

Ha egyazon időben két felhasználó akar hozzáférni ugyanazon a dokumentumhoz, akkor két eset lehetséges, attól függően, hogy engedélyezve van-e a dokumentumok zárolása vagy nincs.

Amennyiben nincs engedélyezve a dokumentumok zárolása:

1, Mindkét felhasználó olvasásra nyitja meg a dokumentumot. Ez semmilyen problémát nem okoz.

2, Az egyik felhasználó szerkesztőmódba rakja, míg a másiknál továbbra is olvasásra van megnyitva. Ekkor, ha megváltoztatja a dokumentumot és elmenti, a változások mindaddig nem látszanak a másik felhasználónál, amíg újra meg nem nyitja. Ha nem vált ő is szerkesztő módba, szintén semmilyen konfliktust nem okoz. Ha szerkesztő módba vált, akkor lásd 3.pont.

3, Mindkét felhasználónál szerkesztő módban van a dokumentum. Ekkor, aki előbb elmenti annak a változtatásai lesznek mentve. Ha a másik felhasználó menteni akar, akkor felugrik egy ablak egy figyelmeztetéssel, hogy a dokumentum megváltozott. Ekkor két lehetőség közül választhat:

a, Elmenti, mint „save conflict document”. Így gyakorlatilag válaszdokumentumként fog elmentődni az eredetire, amit az első felhasználó elmentett, és jelölve lesz, hogy ez egy mentési konfliktus következményeként jött létre.

b, Nem menti el. Ekkor újra meg kell nyitni a dokumentumot és akkor már az érvényes adatokat fogja tudni szerkeszteni.

Amennyiben engedélyezve van a dokumentumok zárolása:

1, Mindkét felhasználó olvasásra nyitja meg a dokumentumot. Ez semmilyen problémát nem okoz.

2, Az egyik felhasználó szerkesztőmódba rakja, míg a másiknál továbbra is olvasásra van megnyitva. Ekkor, aki szerkesztő módba rakta a dokumentumot, az egyúttal zárolja is, ezzel megakadályozva, hogy mások is szerkeszthessék. Ha megváltoztatja a dokumentumot és elmenti (mentés után a zár feloldódik), a változások mindaddig nem látszanak a másik felhasználónál, amíg újra meg nem nyitja. Ha nem vált ő is szerkesztő módba, szintén semmilyen konfliktust nem okoz. Ha szerkesztő módba akar váltani, akkor megjelenik egy

üzenet, hogy a dokumentum megváltozott menet közben, zárja be és nyissa meg újra a dokumentumot. Nem tudja szerkesztő módba tenni, amíg újra meg nem nyitja.

Láthatjuk, hogy jobb megoldás, ha engedélyezzük a dokumentumok zárolását. Azonban az én alkalmazásomban még sincs engedélyezve. Ennek egy igen egyszerű oka van. Mivel az alkalmazás webről is lehet használni és web felől nem lehet zárolni, nincs is értelmezve.
Ha engedélyezve lenne a zárolás az két problémát is felvetne: Először is ahhoz, hogy törölni lehessen egy dokumentumot, zárolni kell azt, ha a zárolás engedélyezve van. Mivel webről nincs zárolás ezért ez így elég problémás lenne. Másodszor, pedig szintén mivel webről nincs értelmezve a zárolás ezért, ha egy zárolt, vagy nem zárolt, de menetközben megváltozott dokumentumot akarunk szerkesztő módba tenni, vagy menteni, a weboldal elhal. Ezt kiküszöbölhetjük, ha az XPage-ünkön konkurens módot (ConcurrencyMode) erőltetvére (forced) állítjuk. Ekkor minden esetben, akár zárolva van egy dokumentum akár nem, ha a weben mentünk valamit ő elmenti. Így viszont már a Noteson belüli zárolásnak sincs túl sok értelme. Egyébként ezt a módot be kell mindenképp állítani, mert mint írtam, nem csak akkor hal el az oldal, ha zárolt dokumentumot akarunk szerkeszteni, hanem akkor is, ha menet közben megváltozott. Ennek következményeként egyszerűen fogalmazva a webes felhasználók élveznek előnyt. Nem kerülhetnek olyan helyzetbe, mint a Notes felhasználók, hogy válasszanak, hogy elmentik konfliktus dokumentumként vagy megnyitják újra szerkesztésre. Én ezt tartom a megfelelő megvalósításnak, mivel így a webes felhasználók nem járhatnak úgy, hogy elhal az oldal, a Notesos felhasználók pedig tudják kezelni ezt a helyzetet. Megoldás lehetett volna még, hogy a ConcurrencyMode-ot „fail”-re állítjuk. Ilyenkor nem menti el a változásokat. De ezt azért vetettem el, mivel a felhasználók egymás dokumentumait úgysem módosíthatják, és egyedül az üzemetetők módosíthatják más felhasználók dokumentumait, ami ritkán fordul elő, így ne kelljen a felhasználónak megnézni, hogy most tényleg elmentette a változást vagy sem. Az üzemeltetők pedig konfliktus helyzet esetén tudják mi a teendő, így a felhasználóknak zavartalanabbá válik a rendszer használata.

7. Összegzés

Az alkalmazást nagyrészt sikerült úgy megvalósítanom, ahogy szerettem volna. Bár néhol szembesülnöm kellett a Notes határaival és a felhasznált új technológia problémáival és megfelelő dokumentációjának hiányával, de sikerült megoldani a felmerülő problémákat és egy működő és véleményem szerint használható alkalmazást készíteni. Mivel minden alkalmazásban, ebben is lehetnek olyan hibák melyekre nem derült fény, mivel egymagam tesztelve nem biztos, hogy minden hiba előjött. De minden új funkció hozzáadása után kipróbáltam azt, így ezzel együtt a régebbiek egyre többször kerültek tesztelésre. Az alkalmazást igyekeztem úgy elkészíteni, hogy plusz funkciókkal könnyen bővíthető legyen és ehhez ne kelljen megváltoztatni a struktúráját.

A diplomamunka írásos részével is többé-kevésbé elégedett vagyok. Kicsit talán kevés benne az elméleti rész és a gyakorlati megvalósításról esik több szó, de mint a bevezetésben is említettem egy konkrét alkalmazáson keresztül akartam bemutatni a Notesos alkalmazásfejlesztést, ezzel is a gyakorlatra helyezve a hangsúlyt. Valójában is több időt töltöttem az alkalmazás fejlesztésével és talán ezért is vagyok nem teljesen elégedett az írásos résszel, de fontosabbnak éreztem a gyakorlati munka szerepét és ezzel a továbbfejlődés lehetőségét. Ezzel egyúttal igazolva, hogy gyakorlatban képes vagyok használni, amit megtanultam, mely tudást nagyrészt magamtól sajátítottam el a rendszerről. Itt mondanék köszönetet témavezetőmnek Dr. Zichar Mariannának, akitől az alapokat tanultam meg, hogy el tudjak indulni, és segítséget kaptam, ahol elakadtam. Valamint Csörgő Józsefnek az National Instruments alkalmazottjának, aki a külső konzulensem volt és gyakorlati tapasztalatával segített.

8. Irodalomjegyzék

1. Séra Tamás – Sudár András: Lotus Notes / Domino, R5 ComputerBooks Kiadói Kft, 2000

2.Unioffice Rendszerház technikai cikkek, A Lotus Notes/Domino története: http://www.unioffice.hu/internet/uoweb.nsf/Default/5869D6EDF6417509C1256CEC004C5243

3. Domino Designer 8.5 Help

4. http://www-10.lotus.com/ldd/ddwiki.nsf