Debreceni Egyetem

Informatikai Kar

Nyilvántartó rendszer Silverlight alapokon

Külsős témavezető:

Keczeli Csaba

CTS Informatika Kft.

Belső referens:

Mecsei Zoltán

Tanársegéd

Készítette:

Szabó Zsolt

Programtervező informatikus

hallgató

Debrecen

2009

Tartalomjegyzék

1. Bevezetés

Tanulmányaim során volt alkalmam betekintést nyerni a Microsoft által készített .NET keretrendszer kisebb szeletébe a C# nyelven keresztül. Ekkor találkoztam a Visual Studio 2005-ös szoftverfejlesztői környezettel, amelyet kiforrottabbnak találtam és persze most is találok, mint a riválisJava nyelvhez szánt és az egyetem során megismert Netbeans és Eclipse szoftverfejlesztői IDE-t. A környezetet idővel a C# nyelvet is meg szeretette velem és ez által a szívem a Microsoft keretrendszere felé kezdett húzni és nem a Sun cég platformjához. Így dolgozatomnak egy olyan témát választottam, amivel egy aktuális témakört boncolgathatok.

A mai világban egyre nagyobb az internet elterjedtsége, átszövi sok 100 millió ember mindennapjait. A kezdeti WEB (World Wide Web) egyoldalúságát fokozatosan felváltotta a jelenségnek mondható WEB 2.0 amely igazából nem is egy verziószámot jelöl hanem internet tartalmak bővülését, az internet kommunikációjának kétirányúvá válását, a közösségi oldalak, blogok megjelenése stb. amely folyamat talán még ma is tart. Ezen változásokkal együtt párhuzamban technológiai elvárások is keletkeztek a tartalom kezelésével és megjelenítésével kapcsolatban, melyeket web alkalmazás platformok kifejlesztésével orvosoltak. Ezek generációs fejlődéséből alakult ki a RIA (Rich Internet Application) fogalom, magyarul „gazdag internetes alkalmazás” melyek piacából a Microsoft is kívánt egy szeletet a Silverlight keretrendszer megalkotásával. A keretrendszer .NET futtatási környezet segítségével elérhetővé teszi a desktop alkalmazások gazdagságának megjelenését, weblap tartalmak vektorgrafikával, animációval és HD videóval való kiterjesztését a böngészőben vagy akár más platformon. A keretrendszer annyira széleskörű eszközfelhozatallal rendelkezik, hogy segítségével lehetőségünk van akár üzleti (Line of Business, LOB) alkalmazások készítésére is. A Silverlight runtime további érdekessége, hogy habár Microsoft termék, támogatja az Internet Exploreren kívül a Mozilla Firefox böngészőjét, illetve létezik Macintosh-on futtatható változata, valamint a fejlesztőcég jelenleg is dolgozik a runtime linuxos változatán, így a Silverlight lehetővé teszi multiplatformos alkalmazások fejlesztését is. Jelenleg egy feltörekvőben lévő rendszerről van szó, amely piaci részesedése ugyan még nem közelítette meg a riválisAdobe cég Flash termékének részesedését. Bízom benne a jelenség a jövőben változni fog és a redmondiaknak sikerül térnyerést elérni a piacon.

Dolgozatommal egy nyilvántartó rendszer megalkotásán keresztül ismertetni szeretném a Microsoft Silverlight 3.0 keretrendszer szolgáltatásait, eszközeit és betekintést nyújtani működésébe a megadott terjedelem megfelelő határain belül. 2.0-ás verzió óta a programozóknak már lehetőségük van Silverlight-os alkalmazásaikat megírni Visual C#, Visual Basic, AJAX, JavaScript, Ruby, és Python nyelv valamelyikén, reprezentálásomban Visual C#-t fogok használni.

2. Silverlight

2.1 Út a Silverlight felé

A keretrendszer használatának bizonyos követelményei vannak a futtató rendszer és a számítógép felé. A számítógépnek minimálisan rendelkeznie kell Microsoft Windows XP operációs rendszerrel, vagy valamelyik utódával (Vista, Windows 7), valamint felhasználóként telepített Silverlight plugin-nal. Fejlesztőként célszerű telepítenünk XP esetén, a .NET Frameworkot amelyet már a Vista, Windows 7 alapból is tartalmaz. Továbbá a Visual Studio 2008 SP1 majd a The Silverlight Tools for Visual Studio telepítése ajánlott az egyszerű kezelhetőség és jobb produktivitás kedvéért, de a fejlesztést meglehet oldani egyéb felületekkel is, amelyek megnehezíthetik életünket a fejlesztés során, ezért ajánlanám a VS 2008 –at, ennek hiányában a Visual Web Developer Express 2008 mely teljesen ingyenesen hozzáférhető az interneten. Továbbá ha tervezünk olyan programok írását, amelyek adatbázisokkal valósít meg kommunikációt, szükségünk lehet SQL Server 2008 Express ingyenes adatbázis kezelő rendszerre. Kiegészítésként a web alkalmazásunk design-jának megalkotáshoz nagyszerű kellék lehet a Microsoft Expression család használta, igény szerint Blend, Design, Web, Encoder. (Dolgozatomban a Expression Blend volt a segítségemre.) Ezek birtokában nyugodtan elővehetjük Silverlight-al foglalkozó könyvünk legfrissebb kiadását vagy éppen a weben fellelhető tutorial-ok valamelyikét, és kísérletezgethetünk a fejlesztéssel.

2.2 Alapok

Egy projekt létrehozásán keresztül fogom bemutatni az alapokat. Tehát a Visual Studio 2008-al (továbbiakban VS) létrehozom Silverlightos alkalmazásom melynek Katalogus3 nevet adtam és egy ASP .NET tesztprojektet választottam egyszerű HTML oldal helyett tesztelésre, amit a VS automatikusan legenerál. Mindenre azért van szükségünk, mert a Silverlight-os alkalmazások plugin keretén belül ASP .NET (Active Server Pages, a Microsoft dinamikus weboldalak generálására alkalmas szerveroldali keretrendszer) vagy HTML oldalakba beágyazottan publikálhatóak. Generálás után ennek megfelelően valójában két darab projekt jött létre az egyik egy ASP.NET-es webprojekt és egy Silverlight-os alkalmazás projekt. A webprojekt több fájlt tartalmaz, ezek között van Katalogus3TestPage.aspx és a Katalogus3TestPage.html teszt oldal is, ezekbe ágyazta be Silverlight alkalmazásomat a VS ugyanazon a módon, így csak az egyiket mutatom be.

```
<div id="silverlightControlHost">
        <object data="data:application/x-silverlight-2,"
type="application/x-silverlight-2" width="100%" height="100%">
              <param name="source" value="ClientBin/Katalogus3.xap"/>
              <param name="onError" value="onSilverlightError" />
              <param name="background" value="white" />
              <param name="minRuntimeVersion" value="3.0.40624.0" />
              <param name="autoUpgrade" value="true" />
              <a
href="http://go.microsoft.com/fwlink/?LinkID=149156&v=3.0.40624.0"
style="text-decoration:none">
                    <img
src="http://go.microsoft.com/fwlink/?LinkId=108181" alt="Get Microsoft
Silverlight" style="border-style:none"/>
              </a>
          </object><iframe id="_sl_historyFrame"
style="visibility:hidden;height:0px;width:0px;border:0px"></iframe>
</div>
```

Az injektálás egy silverlightControlHost régióba lett elhelyezve és tulajdonképpen egy object tag-en keresztül történik 4 kulcs attribútumon keresztül, ahol a

- **data:** Megmondja, hogy egy Silverlight alkalmazásról van szó.
- **type:** Jelzi a minimálisan szükséges Silverlight verziót.
- **width, height: S**zélesség és magasság meghatározás.

és további paraméterekkel. Az object-ben lévő hivatkozás és a kép akkor lesz látható, ha futtatáskor kiderül, hogy számítógépünkre még nem lett feltelepítve a Silverlight plugin. A továbbiakban ezt a hiányosságot orvosolhatjuk a képre való kattintással. Érdekesség, hogy a

Microsoft ellen indított pereknek köszönhetően ez a telepítés már nem történik meg a linkre kattintás után automatikusan, hanem meg kell erősítenünk a telepítési szándékunkat, ráadásul a plugin feltelepítése után az oldal újratöltése sem történik meg automatikusan, hanem ezt a felhasználónak kell kézzel megtennie.

Most pedig nézzük meg VS Solution Explorer-ben a Silverlight (továbbiakban SL) projektek felépítését, ahol alapból két XAML (eXtensible Application Markup Language) kiterjesztésű fájlt fogunk látni egy MainPage.xaml-t és egy App.xaml-t, ezekben fogjuk leírni programunkhoz hozzáadott vezérlők (gombok, szövegdobozok, adattáblázatok, stb.) kinézetét.

XAML

Az objektumokat deklaratív módon leíró nyelv, alapja az XML (eXtensible Markup Language). Egy XAML kódban szereplő elemek azonosíthatóak x: Name=”LayRoot” property által ahol x az adott névtér amiben azonosítjuk, LayRoot pedig maga az azonosító amellyel majd hivatkozhatunk az elemre code behind alól. Elemeket kétféle képpen dekltarálhatunk: Object Element szintaxissal

```
<Button Grid.Row="1" Margin="155,8,145,5" Height="50"
Click="Keres_Button_Click"  >
                <Button.Content>
                    <TextBlock Text="Elkészit" FontSize="20"/>
                </Button.Content>
                 </Button>
```

Ezzel elérve, hogy az elemek Content vagy akár más, tulajdonságát Proprety element-en keresztül ne csak egyszerű, hanem összetett adatokkal tudjunk megadni. A másik módszer Attribute szintaxis, már az XML-ben is fellelhető rövid TAG-es forma:

```
<TextBlock x:Name="loading" Text="Kis türelem" FontSize="30"
HorizontalAlignment="Center" VerticalAlignment="Center" Opacity="0"/>
```

Ahol már a deklarációban beállítjuk a szükséges propertyket, és az esetleges eseményekhez hozzá rendelhetjük az eseménykezelőket.

Xaml-ben találkozhatunk Attached (kapcsolt), Dependency (függőségi), tulajdonságokkal. Attached proprety egy vezérlő Attribute szintaxisában jelenik meg, mely nem az adott elem saját tulajdonsága, hanem az elem szülő vezérlőjéhez kapcsolható. Ezt egy globális tulajdonságnak képzelhetjük el, mely minden vezérlőben állítható.

```
<Button Grid.Row="1" Margin="155,8,145,5" Height="50"…
```

Itt egy Gomb elemdefiníció részlet látható, miben megjelenik egy Kapcsoló tulajdonság (Grid.Row). Ez megmondja a Grid számára, hogy az adott Gombot hol helyezze el a már előzőekben, definiált táblázatban. Ez esetben ez a 2. sor első cellája lesz.(Az indexelés 0-tól kezdődik, az első cella pedig alapértelmezés abban az esetben, ha az oszlopot nem adjuk meg.)

Dependency property-ről általánosan elmondható, hogy egy olyan tulajdonság minek segítségével az adott elem bizonyos tulajdonságát: stílus, érték, default érték, animáció közül akármelyik, honnan származtassa melyik megadott objektumból.

```
<TextBlock x:Name="slidetextfelso"  Style="{StaticResource
alcimek_style}" Height="30"
```

Itt a TextBlock stílusát az alcimek_style stílus objektum fogja leírni, ezt a függőségi tulajdonsággal (StaticResource a példában) állítom be.

SL-ban az események megjelenítése és kezelése külön van választva, ezért a xaml fájljainkhoz tartozik egy úgynevezett „code behind” fájl, ami alapból MainPage.xaml.cs és App.xaml.cs formában jelenik meg. Láthatjuk, hogy négy fájlból kettő-kettő csak a kiterjesztésében tér el, ennek az oka, hogy a SL-ban is megjelent az ún. Partial Class fogalma, amely Részleges osztályt jelent. Jelentőség abban áll, hogy egy osztályt, interfészt, struktúrát, kettő vagy akár több forrás fájlban is definiálhatunk. A tényleges osztály összeállítása a részleges osztályokból fordítási időben következik be. Előnyös megoldás lehet részleges osztályok használata nagy projekteknél ahol fejlesztők egy időben dolgoznak ugyanazon osztályon, illetve a .net keretrendszer fejlesztésben mindenütt ezt a technikát használják akkor, amikor a környezet eszközei/designerei által generált kódot szeparálni szeretnék a fejlesztők által írt kódtól. Használata az osztály definícióban elhelyezett Partial kulcsszóval lehetséges. A fent említett állományainkban írhatjuk le az adott xaml fájlokhoz tartozó események kezelésére vonatkozó tevékenységeinket. Az App.xaml-ben csak alkalmazásszintű erőforrásokat helyezhetünk el pl. stílusok, míg a hozzátartozó App.xaml.cs-ben alkalmazásszintű eseményekre iratkozhatunk fel, mint például az alkalmazás elindítása, leállítása és nem kezelt kivételek. Ezekre az eseményekre az App.xaml.cs-ben lévő App

osztály konstruktorában iratkozunk fel. Így alkalmazás induláskor a vezérlés az alábbi eseménykezelő metódusra kerül:

```
private void Application_Startup(object sender, StartupEventArgs e)
        {
            this.RootVisual = new MainPage();
             }
```

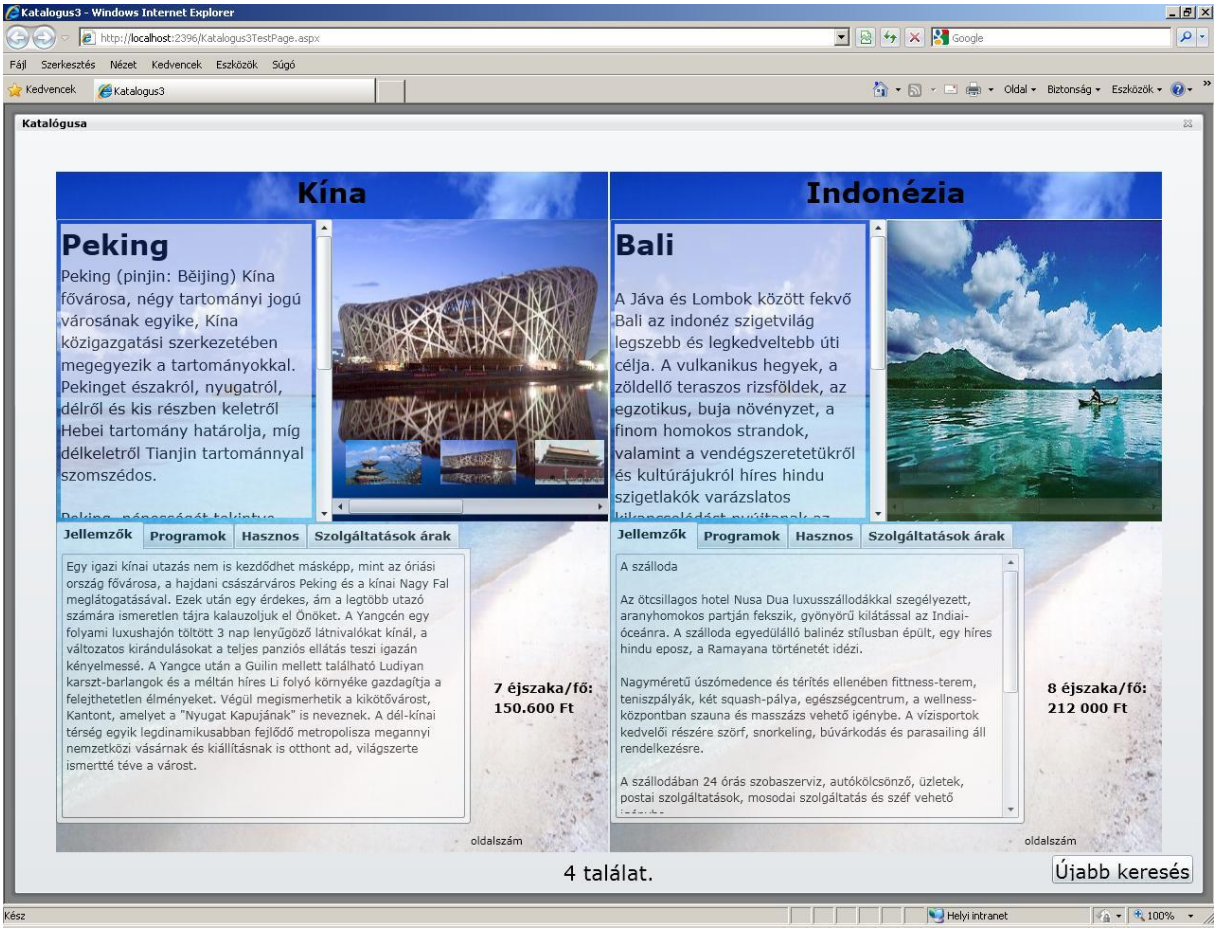
mely a MainPage.xaml.cs fájlban található MainPage osztály egy példányát adja át a gyökér felhasználói felületnek. Ekkor a MainPage konstruktorában lévő InitializeComponent() metódus betölti a MainPage.xaml fájlt és a felhasználó felületen inicializálja az abban található elemeket.

Ha most lefordítom a projektet, akkor a böngészőben üres oldal jelenik meg, mert ugye még semmit nem csináltunk, de észrevehetünk egy változást VS-hez visszatérve. A fordító generált a segéd projektünk ClientBin könyvtárában egy xap kiterjesztésű fájlt, amely tulajdonképpen egy zip fájl és tartalmazza a dll-be fordított projektünk szerelvényeit és AppManifest.xaml fájlt, amely megnevezi az alkalmazásunk szerelvényeit és belépési pontját. Feltehetjük a kérdést, hogyan indul el egy SL alkalmazás? Ha egy felhasználó a böngészőjével egy olyan oldalra téved, amibe egy SL program van beágyazva, akkor ez az alkalmazás letöltődik a szerverről xap fájl formában majd az abban már említett két állomány segítségével az AppManifest.xaml-ből kiderül, mely szerelvény mely osztály tevékenységeivel indul a böngészőben a kód végrehajtása, tehát SL alkalmazások tekintetében egy kliens oldali applikációkról beszélünk.

3. Utazási katalógus

Mint már említettem dolgozatomban nyilvántartó program elkészítésének dokumentálásán keresztül kívánom bemutatni a keretrendszert. Web alkalmazásom egy fiktív utazási iroda utazási ajánlatait fogja megjeleníteni felhasználói felületén, egy kézzel (egérrel) fogható lapozható utazási katalógus formájában. Az adatokat egy utazási ajánlatokat leíró adatbázisból fogja kinyerni és egér segítségével lapozható katalógus formában megjeleníteni. A program el lesz látva minden szokásos funkcióval, amit manapság egy nyilvántartó rendszertől elvárhatunk, gondolok itt a keresések különböző mélységére és számunkra csak a lényeges információk szűkített megjelenítésére. Továbbá elkészítem alkalmazásom párját, mely segítségével menedzselhetem a katalógus mögötti adatbázist.

Applikációm indulásakor egy kereső panelt fogunk látni ahol személyre szabhatjuk utazási katalógusunk a saját céljainknak megfelelően. Opciót találunk a kínálatok szűkítésére: mi alapján mutassunk meg ajánlatokat, országra, kontinensre, vagy adott árkategória szerint nézve. Ha nem töltjük ki a „Hova utazna szívesen?” szekciót, akkor az összes utazási ajánlatot tartalmazni fogja a katalógus, egyébként a kiválasztott ország vagy kontinensnek megfelelően alakul katalógusom tartalma. Keres gomb lenyomását követően „Egy kis türelem!” felirat megjelenésére kerül sor, eközben a háttérben fut a keresésnek eleget tevő lekérdezés mely az adatbázisból kinyeri az aktuális rekordokat és összeállítódik a katalógus szerkezet. Mikor megtörtént az összeállítás és betöltődött a katalógus eseményvezérlés keretében ezt láthatóvá teszem egy új ablak megjelenésén keresztül, ami az adott katalógust fogja tartalmazni és egy új keresés gombot, TextBlock-ot a lekérdezés feltételének eleget tevő rekordok számával.

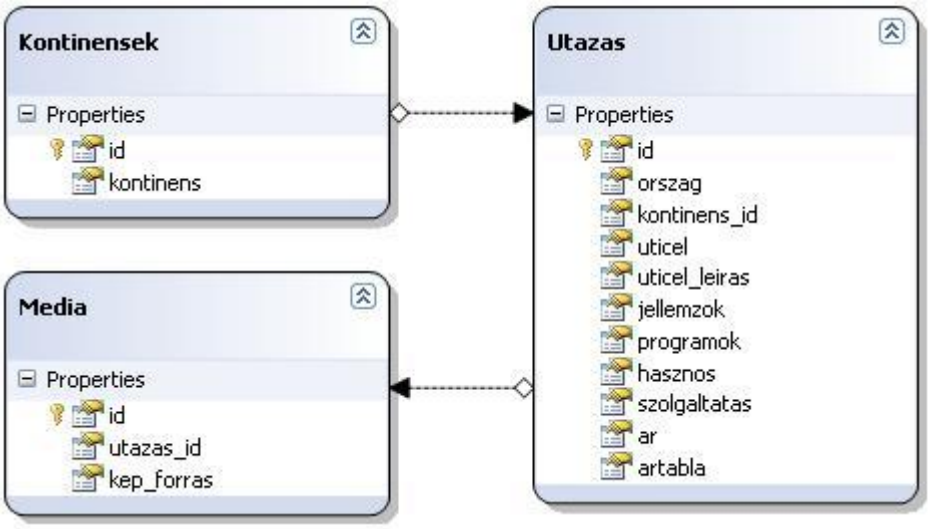


Fentebb a katalógus első oldalát látjuk a képernyőn egy adott első ajánlattal. Katalógusom annyi lapból áll ahány ajánlat eleget tett a keresési feltételnek az utazásokat tartalmazó adatbázisban. Az előállt füzetben egérkattintás segítségével tudunk automatikusan lapozni előre-hátra, vagy a füzet jobb–bal alsó sarkának egérkattintással való megragadása esetén, amit egy kis animáció jelez, lehetőségünk van az oldalt saját magunk manuálisan az egér mozgatásával átlapozni. Programom lapozhatóságának logikáját nem egy beépített Silverlight vezérlő valósítja meg, hanem egy külső nyílt forrásból származó control (SLMitsuControls.dll) felhasználásával valósítottam meg. Ez a Silverlight applikáció a CRUD négyesből csak a Read műveletet valósítja meg select lekérdezések által így fontosnak véltem egy olyan másik Silverlight alkalmazás elkészítését, amely teljessé teszi a körképet a CRUD műveletek tekintetében. A különálló alkalmazás segítségével lehet az utazási ajánlatokat szerkeszteni. A szeparálás célja az volt, hogy a szabadon beépíthető kereső felület koncentráltan a kereséssel tudjon foglalkozni, és gyorsan jelenjen meg az által is, hogy nem hordoz a kereséshez szükségtelen funkcionalitásokat. A különálló Katalógus adatmenedzser alkalmazás egyetlen oldalból áll, amin megtalálható a katalógus egyetlen oldalának mintája,

az előző programban is fellelhető keresőpanel a maga kinézetével és logikájával együtt, továbbá 3 műveleti gomb és egy DataGrid melyben a keresésnek megfelelő sorokat láthatjuk. A DataGridben való kiválasztás által betölthetjük az adott elem tulajdonságait a minta oldalba és kedvünk szerint szerkeszthetjük, módosíthatjuk, majd menthetjük.

3.1 Adatbázis

Microsoft Sql Server 2005 Express adatbázis-kezelőt választottam a programom, adatbázis kiszolgálójának. Magát az adatbázist egy utazásokra vonatkozó információkat tartalmazó (Utazas) tábla és egy képek elérési útjait tartalmazó (Media) tábla, továbbá egy az összes kontinenst tartalmazó (Kontinensek) tábla alkot. A tábla Amerikát mint kontinens Észak-Amerika, Dél-Amerika és Közép-Amerikára bontva tartalmazza. Az Utazas és Kontinensek tábla kapcsolatát az Utazas tábla kontinens_id mezőjének megadott külső kulcs hivatkozása valósítja meg a Kontinensek tábla id mezőjére. A Media és az Utazas relációk közötti kapcsolatot a Media tábla utazas_id mező külső kulcs megszorítása biztosítja, ezáltal csak olyan adatokat tartalmazhat, mely adatok szerepelnek az Utazás tábla Id mezőjében.



A három táblában szereplő id elsődleges kulcsot alkot típusa Integer32, automata növekményes 1 növekménnyel, a külső kulcs megszorítással rendelkező kontinens_id és utazas_id típusa is Integer32. Táblákban a további megjelenő mezők mind rendre NVarChar (National Character Varying) típusúak az ar mezőtől eltekintve, melynek típusa numerikus. NVarChar egy karaktereket tartalmazó adattípus UNICODE karakterkódolással, maximum 4

Kb. hosszon. A táblák egyes NVarChar mezőinek a hosszát az előbb említett maximális hossztól eltérően adtam meg, mint például az Utazas tábla ország mezőjénél, ahol az előforduló karakterek számát 40 karakterben maximalizáltam Egy NVarChar mező a szövegek tárolására mindig csak a szükséges helyet foglalja le, szemben a fix hosszúságú NChar szöveges mezővel.

Utazások táblámnak 11 oszlopa van.

- Id: Elsődleges kulcs.
- Orszag: Utazások célországa.
- Kontinens_id: Utazás országának kontinensét leíró Kontinensek sorára való hivatkozás, vagyis külső kulcs Kontinensek tábla id-jára hivatkozik.
- Uticel: Utazás országának valamely földrajzi helye, városa.
- Uticel_leiras: Bemutató szöveg az úti célról.
- Jellemzők: Elszállásolásról szóló információ.
- Programok: Utazáson végezhető szervezett/szervezetlen tevékenységek.
- Hasznos: Célországgal kapcsolatos közérdekű információk.
- Szolgaltatas: Utazással kapcsolatos szolgáltatások és árai.
- Ar: Utazás alap ára. (Felhasználótól elrejtett adat)
- Artabla: Felhasználó számára látható utazás ára kiegészítő szöveggel: pl. 125.000 Ft 8 éjszaka/fő

Media tábla három oszlopa:

- Id: Elsődleges kulcs.
- utazas_id: Egy külső kulcs, az Utazas tábla id-jára hivatkozik.
- kep_forras: Egy kép URL-je.

Kontinensek tábla két oszlopa:

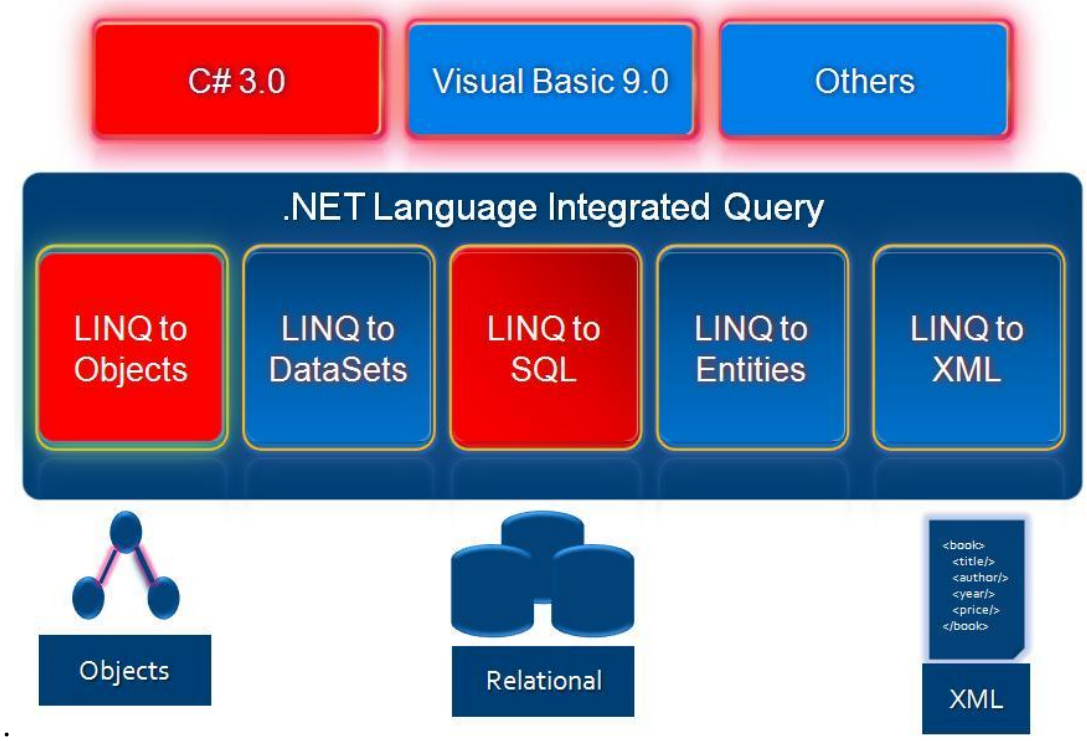
- Id: Elsődleges kulcs.
- Kontinens: Földünk egy adott kontinense.

3.2 Alkalmazás és az adatbázis kapcsolata

Egy adatbázissal dolgozó alkalmazás számára az adatbázisban megjelenő típusokhoz hozzá kell rendelni az objektumorientált nyelv megfelelő típusait, ezt .NET 3.0-ban megjelenő LINQ (Language Integrated Query) technológiával oldom meg.

3.2.1 LINQ-A nyelvbe ágyazott lekérdező keretrendszer

Egy technológia mely az objektum és nem objektum orientált adatforrások kezelését egyszerűsíti le, továbbá egységesíti is, akár XML hierarchiáról, relációs adatbázisról vagy éppen egy objektumorientált IEnumerable, IEnumerable<T> interfészt megvalósító adatforrásról legyen szó.



Használatához egyszerűen az adott osztályhoz kell adni a System.Linq névteret. Két lekérdező metódusforma áll rendelkezésünkre a LINQ-ban:

```
string[] Nevek = {"Pista", "Zsolt","Hugó","Dóra","Emil" };
            var query = from c in Nevek
                        where c.Length > 4
                        select c;
```

A fenti lekérdezés a Nevek stringeket tartalmazó tömbből visszaadja a query változóba a négy betűnél hosszabb neveket. Ez a lekérdezés forma nagyon hasonlít az általános SQL query szintaxisához, így ezt Query kifejezés nevet kapta. A másik:

```
string[] Nevek = { "Pista", "Zsolt", "Hugó", "Dóra", "Emil" };
            var query2 = Nevek.Where(q => q.Length > 4)
                              .Select(q => q);
```

A query2 lekérdezés teljesen ekvivalens az előző lekérdezéssel, tehát a négy betűnél hosszabb neveket adja vissza. Ez már inkább metódusra formára emlékeztet ezért a Metódus alapú lekérdezés nevet kapta. Az első könnyebben olvasható, de végeredményben a LINQ fordításkor a Query kifejezéseket először Metódus alapúra lekérdezésekké formálja, majd azokat fordítja. A fent szereplő select, where kulcsszavak kiegészítő metódusokat jelölnek, melyek kiterjesztik a Nevek objektum képességeit. Paraméterük a 2. esetben lambda kifejezés, ezek segítségével akár kódot is tudunk átadni paraméterül. A lambda kifejezések szintaxisa:

paraméter => kifejezés.

`Nevek.Where(q => q.Length > 4)` jelentése: Válogasd ki azokat a q-kat amikre igaz, hogy q hosszabb mint négy. q típusa nincs megadva, ugyanis ez a Where metod alapján határozódik meg ami egy Nevek lesz, de ezt van lehetőségünk explicit úton is megadni `Nevek.Where((Nevek)q => q.Length > 4)` formában.

További újdonságnak számít C# 3.0–ban a var kulcsszó használata, segítségével implicit módon kapja a változó a típusát. Nyelvtanilag a var veszi a kifejezés jobb oldalán a kifejezés kiértékelés végén előálló típust és azt adja a változó erős típusának. A két példában szereplő query, query2 típusa tehát IEnumerable<string> lesz. Így, bár típus nélküli változó, az első felhasználáskor egy konkrét erős típusra lesz kényszerítve, és szemben a többi típus nélküli változót kezelő nyelvvel, ez a változó típuskonvertáló műveletek nélkül nem alakítható át másik típussá szabadon.

Az alkalmazásom relációs adatforrással dolgozik, így az ennek megfelelően LINQ to SQL osztályt, használok. A Visual Studio egy támogatása a Linq to SQL felé, hogy képes designer felületen keresztül létrehozni az adatbázis elemek objektumorientált környezetben megjelenő elemeit. Mindezt a Server Explorer-ben megjelenő adatbázis elemeknek csupán a designer felületre való húzásával elérhetjük. Annak ellenére, hogy csak a táblákat tudjuk a

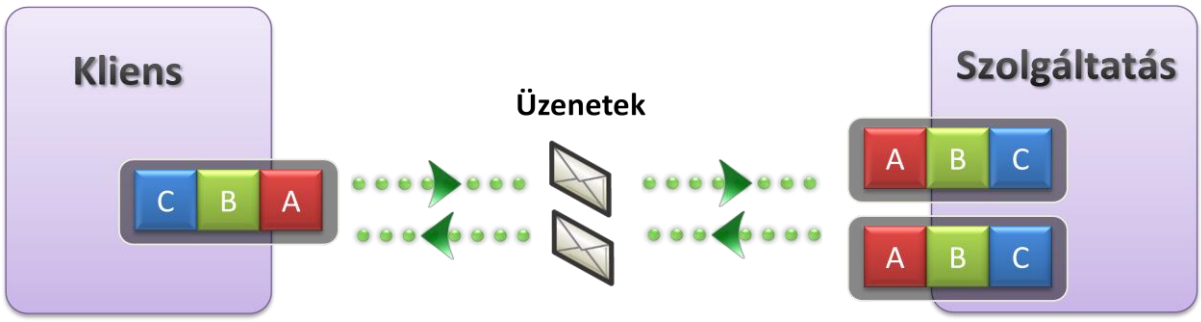
designer felületére húzni, a környezet a relációkat is felismeri, és automatikusan beépíti a modellbe.

Most hogy megvan az adatbázis és program közötti megfeleltetés a LINQ to SQL által, muszáj megjegyezni, SL programom egy kliensalkalmazás, ezért valamilyen módon el kell érnie a tényleges fizikai adatbázist, ezt web szolgáltatáson keresztül éri el, melyet WCF (Windows Communication Foundation) valósít meg.

3.2.3 Windows Communication Foundation (WCF)

A WCF .NET3.0-ban megjelent technológia mely egy egységes programozási modellt nyújt szolgáltatás orientált alkalmazások számára, így teljesen mindegy mi a kommunikáció alapja (TCP/IP vagy HTTP, esetleg egyéb), a fejlesztő ugyanabban a környezetben dolgozhat ezáltal a fejlesztők hatékonyabban és kevesebb kód megírásával fejleszthetnek elosztott alkalmazásokat, mint a régebbi technológiák esetén.

WCF esetén két szerepről beszélhetünk, **szolgáltatás** (esetemben a szolgáltatás szerepe a webszerver-é) és **kliens** (a kliensé pedig a webalkalmazás szerep) melyek egymással üzeneteken keresztül kommunikálhatnak.



A végpontok három dolgot fognak össze egy **Address-t,** mely egy URL minek segítségével elérjük a szolgáltatást, egy **Binding-t** ami definiálja a kommunikáció típusát (TCP/IP vagy HTTP, esetleg egyéb), továbbá egy **Conrtact-ot** mely a szolgáltatás interface-t írja le.

A WCF több kötést ismer BasicHttpBinding, NetTcpBinding stb., mivel esetemben csak a HTTP alapú érdekes, ugyanis a Silverlight egyedül a BasicHttpBinding-et támogatja, ami szöveges vagy XML formátumú üzenetkódolást definiál.

Szolgáltatásszerződés egy olyan osztály interfész, mely definíciója előtt szerepel a [ServiceContract] attribútum. Ez az osztály fogja publikálni szolgáltatásaink műveleteit. Katalógusomban Műveletszerződéseket alkalmazok, amik tulajdonképp metódusok [OperationContract] attribútummal ellátva. Ezen elemekkel összeállított szolgáltatást publikálni kell, történhet winform, wpf stb. alkalmazásokon keresztül, vagy egy IIS (Internet Informaiton Servce) webszerveren keresztül.

Hogyan épül ez fel az alkalmazásomban? Szolgáltató oldalon, vagyis a webes projektben egy szolgáltatást leíró osztályt hozok létre, mely egy DataContext példányt és Műveletszerződéseket tartalmaz. A DataContext-re pontosabban a ZsoltiDataContext-re azért van szükségünk, mert ő kezeli az adatbázis kapcsolatot: megnyitja, lezárja, és kezeli az adatbázisból lekérdezés által visszakapott adatokat. A műveletszerződések pedig leírják az adatbázis műveleteket: select, insert, update, delete. A fentieket szemléltetve:

```
namespace Katalogus3.Web
{
    [ServiceContract(Namespace = "")]
    [AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
    public class AdatforrasDataContext
    {
        AdatforrasDataContext adt = new AdatforrasDataContext();

        [OperationContract]
        public List<Utaza> GetKontintensUtazas(string s)
        {
            List<Utaza> result = new List<Utaza>();
            result = (from c in adt.Utazas
                      where c.Kontinensek.kontinens.Equals(s)
                      select c).ToList();
            return result;
        }…
```

Kliens oldalon csupán annyit kell tennünk egy szolgáltatás elérése érdekében, hogy a kliens szolgáltatás referenciáihoz hozzáadjuk a szerver oldalon elkészített service referenciát, ezáltal generálódott egy kliens oldali proxy osztály és a szolgáltatás beállításit tartalmazó ServiceReferences.ClientConfig fájl a SL projektben. A config fájlban találjuk meg a szolgáltatás biztosító hoszt URL-jét. A generált proxy osztály szerepe, hogy rajta keresztül tudjuk elérni a szolgáltatásban tartalmazott műveletszerződéseket. Adott osztályban, hol szeretnénk feldolgozni az adatbázis elemeit ott deklarálni kell a generált proxy osztályt (AdatforrasServiceClient) és ezen osztályon keresztül hivatkozva elérhetjük a szolgáltatás lekérdezéseit. Ezen műveletszerződéseket aszinkron metódushívásokon keresztül érjük el, az

alkalmazás futása közben a lekérdezés egy mellékszálon történik nem várakoztatva ezzel a program futtatását, ugyanakkor feliratkozhatunk az adott aszinkron metódus Completed eseményére, mely segítségével akár alkalmazás akár a felhasználói interakciót is várakozásra kényszeríthetjük.

3.3 Katalógusom UI felülete

Applikációm egy főablakból és egy gyermek ablakból áll. A főablakban egy keresőpanel kapott helyet, míg a gyermekablakban egy „új keresés” feliratú gomb, egy TextBlock mely a keresés feltételének eleget tevő találatok számát szemlélteti és egy lapozhatóságot megvalósító UCbook külső vezérlő, mely oldalának felépítését az Oldal.xaml fájlban építem fel. Érthetőség kedvéért kívánom megjegyezni, hogy a katalógus egy oldalát fentről lefele haladva fogom bemutatni, kifejteni milyen elemből építem fel és mi az adott SL eszköz jellemző tulajdonságai és mikor használatos (ezt minden egyes tool első előfordulásánál teszem a továbbiakban nem részletezem), hogyan jelenik meg ez az adott tool az applikációmban, majd ezek után a kereső panel szerkezetét is bemutatom, persze mindezt nem teljes részletességgel, a terjedelem határainak megfelelően.

3.3.1 Egy oldal felépítése

Mikor létrehozunk egy projektet, létrejön egy „Usercontrol”, melyben fogjuk elhelyezni a web oldalon megjelenítésre szánt vezérlő elemeinket, az egyes Usercontrol-nak jól formázottnak kell lenni, csak egyetlen gyökérelemet tartalmazhat ez általában, Grid, Canvas, StackPanel elrendezés vezérlő valamelyike. Ezen vezérlők közös tulajdonsága, hogy több gyermek elemmel rendelkezhetnek, így konténer vezérlőknek is hívjuk. Border vezérlőt is elrendezés vezérlőnek hívják bár csak egy gyermek eleme lehetséges, igaz ez Object Element szintaxissal való deklaráció esetén könnyen megkerülhető. Ezen elrendezés vezérlők további lényeges tulajdonsága, hogy tetszés szerint egymásba ágyazhatók.

Canvas vezérlő

A legegyszerűbb elrendezés vezérlő, elemek elhelyezésének finomhangolásra való, bonyolult RIA felületek megoldására inkább a másik három elrendezés vezérlő alkalmas. Az elemeket helyét abszolút pozicionálással koordinátákkal kell megadni. A Canvas-an elhelyezett vezérlők alap pozíciója a (0,0) pont, ezen változtatni a Canvas.Top (Canvas tetejétől való behúzás a megadott egységgel), Canvas.Left (Canvas bal oldalától való behúzás a megadott egységgel) Attached property-ken keresztül lehet minden egyes vezérlőnél megadni. Lehetőségünk van az egymás felett elhelyezett objektumok közül kiválasztani melyik jelenjen meg a kettő közül, ezt a Canvas.Zindex tulajdonságán keresztül lehet. A szabály, hogy az egymáson elhelyezett elemek közül mindig a nagyobb Zindex-ű elem látszik.

Stackpanel vezérlő

Az elemeket egyszerűen egymás alá vagy mellé helyezi, attól függően, hogy az Orientation tulajdonságában Vertical vagy Horizontal érték szerepel. Alapbeállítás az egymás alá helyezés. Függőleges orientáció esetén a panelben elhelyezett elemek kitöltés tulajdonsággal vízszintesen fogják kitölteni a konténervezérlő által rendelkezésre álló területet, vízszintes orientáció esetén ez a kitöltés függőlegesen történik.

Grid vezérlő

A leghasználhatóbb elrendezés vezérlő, sorok és oszlopok definiálásával végeredményben egy láthatatlan rácsszerkezetű táblázat hozható létre, mely celláiban akár egynél több vezérlő is elhelyezhető. A sorok és oszlopok definiálásakor ha nem adunk meg méreteket akkor minden sor és oszlop azonos méretű lesz. Fejlesztői átláthatóság jegyében lehetőségünk van a rácsszerkezet megjelenítésére a ShowGridLines property true-értéknek megadásával. Ugyanakkor, a kész alkalmazás megjelenését rontja, így az alkalmazás elkészültével érdemes a rácsszerkezet elrejtése. A vezérlőben elhelyezett elemeknek explicit megadható, hogy a Grid mely cellájában helyezkedjen el, ezt a Grid.Row és a Grid.Column attached property-k beállításával lehet. Fontos megjegyezni a sorok és oszlopok indexelése a nullától kezdődik. Az előbbi property-k megadásának hiányában a vezérlők alapbeállításként a (0,0) indexű cellában fognak elhelyezkedni. Grid.RowSpan és a Grid.ColumnSpan Attached property-k megadásával megadható, hogy az adott vezérlő a Grid egy celláján túlmenően mely további celláit töltse ki.

Katalógusom egy oldalának összetettsége miatt egy Grid elrendezés vezérlőt választottam.

```
<Grid ShowGridLines="False" Visibility="Visible" >
      <Grid.ColumnDefinitions>
            <ColumnDefinition/>
             …
      </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition Height="40" />
                <RowDefinition/>
                <RowDefinition/>
                 </Grid.RowDefinitions>
      </Grid>
```

Ezzel a kis XAML kóddal egy Grid-et definiáltam melynek háttere fehér és éppen a rácsvonalai nem láthatóak, maga a Grid pedig látható. A következő sorban egy Tag-en keresztül beállítottam az oszlopok számát oszlopdefiníció segítségével majd egy sorokat definiáló Tag-en keresztül a sorok számát, esetemben 4 oszlopról és 6 sorról van szó.

A katalógus oldala rendelkezik fejléccel, mely tartalmazza az ajánlat országát, ezt egy TextBlock elem valósít meg.

TextBlock

Abban az esetben használatos ez az eszköz, ha statikus szöveget szeretnénk megjeleníteni az UI (User interface) adott területén, karakter bevitelre nem alkalmas. Adott karakterek megjelenítését Text tulajdonságán keresztül állíthatjuk be. Lehetőségünk van tartalmának formázására, betűcsaládok közötti választásra, betűméret megadására, betűstílus megadásra, kiemelésre, szövegigazításra, explicit sortörésre. Beállításuk tulajdonságokon keresztül történik kivétel az explicit sortörés, amelyet a szövegrészben kell elhelyezni persze nem a megszokott HTML tag-eken keresztül, hanem speciális <LineBreak/> kulcsszóval. Egy TextBlock-on belül a karaktereknek adhatunk egymástól különböző stílust, ekkor használnunk kell a TB-n belül egy Run objektumot, mely kezdő és záró elemén belül az eddig beállított stílust felüldefiniálhatjuk. Az Oldal osztályban a TextBlock-ok Text tulajdonságát „Binding” kulcsszó segítségével kötöm az adatbázisból kinyert utazások megfelelő mezőihez. Ez egy egyirányú OneWay kötés ugyanis ebben az applikációban a célom csak megjelenítés, módosítás törlés nem érdekes jelen esetben.

Programomban a fejlécét az alábbi kód reprezentálja:

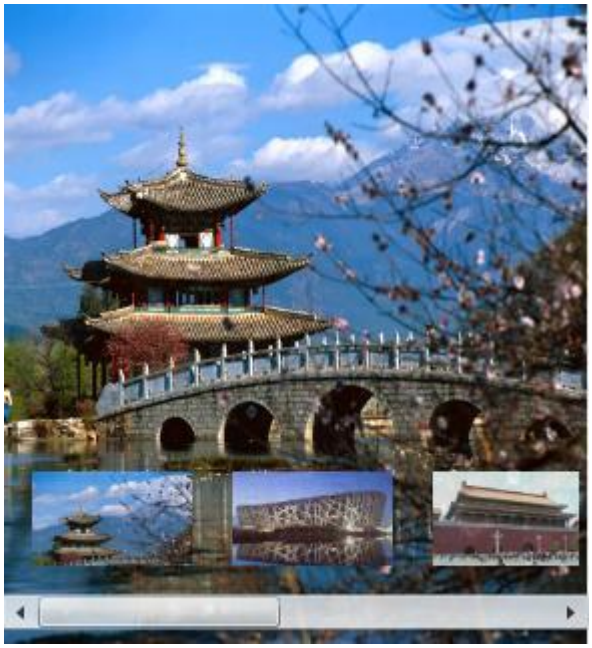
```
<TextBlock Name="Fejlec" Grid.Row="0" Grid.ColumnSpan="4" FontSize="30" .../>
```

Fejlécemet alkotó TB-nak a „fejléc” nevet adtam, a Grid 0. sorában és a 0. oszlopában helyezkedik el, a kódból kitűnik, hogy a 0. sorban van elhelyezve, de hogy melyik oszlopban az nem. Ez azért lehetséges, mert a Grid-ben elhelyezett vezérlők alapértelmezetten a 0. sor 0. oszlopába kerülnek és ha egyébként is ide szeretnék elhelyezni egy elemet akkor a Grid.Row, Grid.Column Attached property-k beállítása elhagyható. Jelen esetben a Grid.Row=”0” szerepe csupán a szemléltetés. A Grid.ColumnSpan tulajdonság értékével kiterjesztettem a cella nagyságát az összes oszlopra. Szöveg tulajdonságát direkt nem tüntettem fel, elől-járóban annyit mondanék egy adatbáziskötést fog megvalósítani, amit a későbbiekben pontosan részleteznék.

Elhelyezek egy rövidke bemutató leírást az adott oldal uticéljáról, nevezetességek és egyéb turisztikai szempontból fontos látványosságairól. Két TextBlock-al fogom megvalósítani mindezt. Programom jellege miatt kevés a rendelkezésre álló hely a bemutatásra, így a leírás bizonyára kitölti nagy többségben túl is szárnyalja a rendelkezésre álló helyet, ezért a két TextBlock-ot egy ScrollViewer vezérlőbe ágyazva fogom megjeleníteni, így gördítő sáv segítségével már lesz alkalmam a teljes szöveget áttekinteni. Ezen eszköz Content tulajdonságába helyezhetjük el a megjeleníteni kívánt elemet. Figyelembe véve azt a tényt, hogy a ScrollViewer vezérlőnek Content tulajdonságának csupán egy gyermek eleme lehet, és mi ennél több eszközt kívánunk elhelyezni ScrollViewer elemben akkor ezt egy konténervezérlőbe (Canvas, Grid, StackPanel) való ágyazás segítségével tehetjük meg, mint ahogy tettem is ezt. Beállíthatjuk a függőleges, vízszintes gördítősávok láthatóságát, aktivitását, passzivitását, és a szokásos méretezési, margó, láthatósági tulajdonságokat. SVR-m Content tulajdonságának a tartalma egy StackPanel vezérlő és ebben elhelyezve két TextBlock, amelyet az alábbi kód reprezentál:

```
<ScrollViewer Grid.Row="2" Grid.ColumnSpan="2" Background="Aqua"
VerticalScrollBarVisibility="Visible"  BorderThickness="1,1,1,0" >
                <ScrollViewer.Content>
                    <StackPanel>
                        <TextBlock  TextWrapping="Wrap" FontSize="30"
FontWeight="Bold" FontFamily="Portable User Interface"  Text="{Binding
uticel}"/>
                        <TextBlock TextWrapping="Wrap"  FontSize="16"
Text="{Binding leiras}"/>
                    </StackPanel>
                </ScrollViewer.Content>
            </ScrollViewer>
```

Vezérlőmet a Grid 2. sorában és első két oszlopában helyeztem el.



A következő egysége oldalamnak a képeket kezelő Grid amelynek háttere szolgál az utazásokhoz tartozó képek megjelenítésére. Mivel több képet kívánok minden egyes utazáshoz megjelenítésre felajánlani, de a Grid háttere egyszerre csak egy képem megjelenítésre szolgál, így készítettem egy funkciót, amely segítségével kiválaszthatjuk, melyik képet lássuk éppen a Grid hátterében. Ez a funkciót egy ScrollViewer-el oldom meg, mely tartalmát dinamikusan Code Behind-ból töltöm fel: Az oldal konstruktora meghívja a HozzaadKepek() eljárást, ami szervizkliensen keresztül feliratkozik GetkepekCompleted eseményre majd paraméterként az aktuális utazás id-jével GetKepek aszinkron metódussal lekérdezést intéz a szolgáltató fele. A GetKepekCompleted eseménykezelőben a lekérdezés eredményének létrehozok egy képeket tartalmazó listát és tartalmát a lekérdezésből kapott lista bejárásánál adom meg, listához fűzök egy kép objektumot egy Uri-n (uniform resource identifier-egységes forrás azonosító) keresztül megadom a kép forrását. Végső feladatként feliratkozom a MouseLeftButtonDown eseményére.

Mivel egy utazáshoz több kép tartozhat és tudjuk, hogy a ScrollViewer Content-jének csupán egy eleme lehet, ezért StackPanel konténervezérlőt alkalmazok vízszintes orientációval. Továbbá cél volt, hogy ez a funkció lehetőség szerint ne rontsa az éppen megjelenített kép tartalmának élvezhetőségét. Animálás segítségével oldottam meg ezt a feladatot.

```
<Grid x:Name="Kep_Grid" Grid.Row="2" Grid.Column="2" Grid.ColumnSpan="2">
                <Grid.RowDefinitions>
                    <RowDefinition/>
                    <RowDefinition Height="100"/>
                </Grid.RowDefinitions>

                <ScrollViewer x:Name="Kep_ScrollViewer" Opacity="0.2"
ScrollViewer.VerticalScrollBarVisibility="Disabled"
```

```
ScrollViewer.HorizontalScrollBarVisibility="Visible"
VerticalAlignment="Bottom" HorizontalContentAlignment="Center"
UseLayoutRounding="True" VerticalContentAlignment="Center"  Grid.Row="1"
HorizontalAlignment="Center" Margin="0,0,0,8" BorderThickness="0"
MouseEnter="MouseInto" MouseLeave="Mouseout" >
                    <StackPanel x:Name="Kep_StackPanel"
Orientation="Horizontal">

                    </StackPanel>
                     </ScrollViewer>…
```

Tehát létrehoztam egy Grid vezérlőt 2 sorral, melynek a második sorában helyeztem a Kep_ScrollViewer képkiválasztó funkciómat összefogó vezérlőt és méretezési, elhelyezési tulajdonságai mellett beállítottam az alap láthatóságát Opacity-n keresztül 20%-ra amit animáción keresztül fogok 85%-ra állítani abban az esetben, amikor az egér mutatója belépett a ScrollViewer területébe és vissza 20%-ra mikor elhagyta azt. Tehát feliratkoztatom vezérlőmet a MouseEnter és MouseLeave eseményekre, melyek bekövetkezésekor meghívódik a MouseInto illetve Mouseout eseménykezelő, amik meghívják az adott eseményhez beállított animációt:

```
private void MouseInto(object sender, System.Windows.Input.MouseEventArgs
e)
        {
            VisualStateManager.GoToState(this, "Highlighted", true);
        }
```

Visual State

Amikor ScrollViewer-em éppen 20%-ban látható alkalmazásom felületén vagy éppen 85%-ban, elmondható hogy a vezérlő megjelenítése egy bizonyos állapotban van, ezekre az állapotokra használja a Silverlight a Visual state jelzőt. Ezen állapotváltozásokhoz rendelhetnünk animációkat. VisualStateManager objektumban definiálhatunk különböző állapot csoportokat VisualStateGroup elemeken keresztül, melyekbe jellegét tekintve az egymáshoz hasonló VisualState objektumokat célszerű csoportosítani. A Silverlight API–ban némely kontrolnak van előre definiált vizuális állapota, mint például a gomb vezérlőnek: a CommonStates csoporton belül a Normal (alap), MouseOver (az egérmutató a gomb felett van), Pressed (megnyomott) állapota.

Mindezt azért tartottam fontosnak megjegyezni, mert az én animációmban is szerep jutott az állapotok használatának. A fenti kódsorban is kiderül egy állapot váltásról van szó, ahol a cél állapotunk a Highlighted state. Eme változás hatására pedig lefut a beállított animáció.

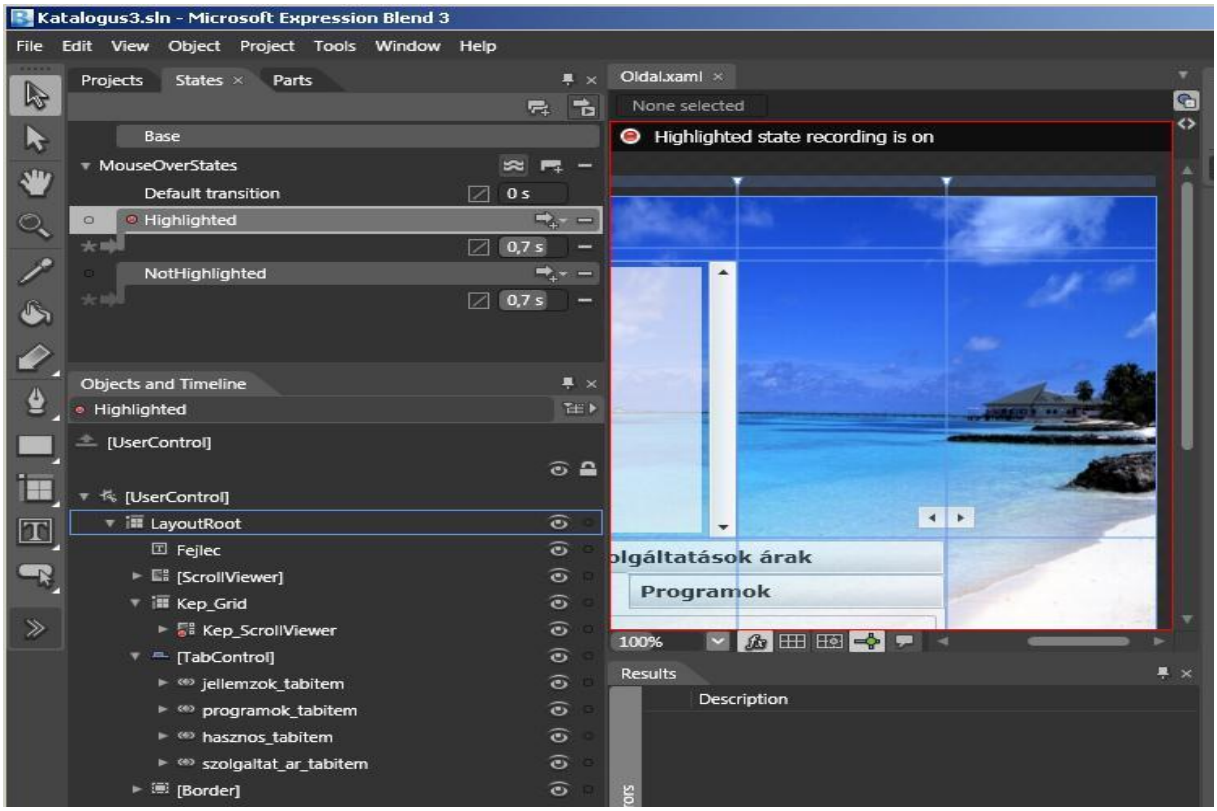
Animációk

Silverlight-ban kétféle típusú animációt különböztethetünk meg, from/to animáció és key frame animáció. A from/to a legegyszerűbb animációfajta egy objektum cél tulajdonságát animálja, egy megadod kezdő értékből egy végértékbe lineáris átmeneten keresztül. Key frame animációk összetettebbek, ugyanis itt nem csak két értéket tudunk megadni, amelyek között végrehajtódjon az animáció, hanem bizonyos időpontokhoz tudunk rendelni értékeket, melyeket vegyen fel az animálandó objektum valamely tulajdonsága. Emellett lehetőség van arra, hogy ezek között az időpontok között az animálás ne egy lineáris átmeneten keresztül menjen végbe, hanem görbét leíró átmeneten keresztül.

Animációkat a Silverlight Storyboard vezérlőkön keresztül implementál, ezek tartalmazhatnak egy vagy több gyermek animációs vezérlőt. A Storyboard rendelkezik az animáció lejátszására, megállítására és a lejátszás szüneteltetésére való képességgel. Az alábbi Silverlight animáció vezérlőket tartalmazhatja:

- ColorAnimation és ColorAnimationUsingKeyFrames: olyan vezérlő tulajdonságok animálására hol a tulajdonság színkezelésen alapszik. Pl.: SolidColorBrush
- DoubleAnimation és DoubleAnimationUsingKeyFrames: olyan vezérlő tulajdonságok animálására hol a tulajdonság integer és double számokon alapszik. Pl.: With (szélesség), Opacity (láthatóság), stb.
- PointAnimation és PointAnimationUsingKeyFrames: olyan vezérlő tulajdonságok animálására hol a tulajdonság X, Y koordináta pontokon alapszik.
- ObjectAnimationUsingKeyFrames: vezérlők objektumhoz kapcsolódó tulajdonságainak animálására. Pl. Fill (kitöltés)

Egy Key frame típusú DoubleAnimationUsingKeyFrames animációt használtam megoldásom megvalósítására, ennek elkészítését nagyban megkönnyíti a Microsoft Expression Blend használata, mi által pusztán kattintgatással össze tudjuk állítani animációnkat XAML-ben való aprólékos kódolás helyett.



A Kep_ScrollViewer animálandó objektumom mivel nem rendelkezik előre definiált vizuális állapottal ezért létrehoztam egy Highlighted és NotHighlighted állapotot a MouseOverStates csoportban. Majd a Highlighted state-re kattintva az XAML megjelenítő tetején jelzi nekünk a Blend, hogy animáció felvételt végzünk Highlighted állapot számára, amit a képen is látható píros pont és mellette szöveg szemléltet. Ekkor a Kep_ScrollViewer láthatósági tulajdonságának megadom 80%-os értéket. Az animáció állapotváltozáshoz van kötve, ezáltal megadhatjuk, hogy az adott animáció melyik állapotból melyik állapotba történő változásakor fusson le. Esetemben a láthatóság növelést eredményező animációt a minden állapotból Highlighted állapotba történő változás esetén futtatom 0.7 másodperces hosszal. A NotHighleghted esetben is hasonlóan történik az animáció azzal a különbséggel, hogy ott leveszem az Opacity-t 15%-ra. Ezek megtétel után megkaptam a következő Blend által generált kódot, amely reprezentálja az elhangzottakat.

```
<VisualStateManager.VisualStateGroups>
                <VisualStateGroup x:Name="MouseOverStates">
                    <VisualStateGroup.Transitions>
                        <VisualTransition
GeneratedDuration="00:00:00.7000000" To="Highlighted"/>
                        <VisualTransition
GeneratedDuration="00:00:00.7000000" To="NotHighlighted"/>
                    </VisualStateGroup.Transitions>
```

```
                    <VisualState x:Name="Highlighted">
                        <Storyboard>
                            <DoubleAnimationUsingKeyFrames
BeginTime="00:00:00" Duration="00:00:00.0010000"
Storyboard.TargetName="Kep_ScrollViewer"
Storyboard.TargetProperty="(UIElement.Opacity)">
                                <EasingDoubleKeyFrame KeyTime="00:00:00"
Value="0.85"/>
                            </DoubleAnimationUsingKeyFrames>
                        </Storyboard>
                    </VisualState>
                    <VisualState x:Name="NotHighlighted">
                        <Storyboard>
                            <DoubleAnimationUsingKeyFrames
BeginTime="00:00:00" Duration="00:00:00.0010000"
Storyboard.TargetName="Kep_ScrollViewer"
Storyboard.TargetProperty="(UIElement.Opacity)">
                                <EasingDoubleKeyFrame KeyTime="00:00:00"
Value="0.15"/>
                            </DoubleAnimationUsingKeyFrames>
                        </Storyboard>
                    </VisualState>
                </VisualStateGroup>
            </VisualStateManager.VisualStateGroups>
```

A következő elem az adott utazásról fog hasznos információkat megjeleníteni különböző csoportokba sorolva: Jellemzők; Programok; Hasznos; Szolgáltatások, árak. A csoportok megvalósítását TabControl segítségével értem el, melyben a fent említett csoportoknak megfelelően létrehoztam TabItemeket.

TabControl

Használatos, ha adott területen több különböző szerkezetet szeretnénk kialakítani vezérlőkből vagy a vezérlők megjelenített tartalmát szeretnénk elszeparálni egymástól. A TabItemek létrehozásával hozhatjuk létre a TabControl különböző paneljeit, melynek a Header tulajdonságán keresztül adhatjuk meg a panel fejlécét. Egy TabItem-en csupán egyetlen vezérlő helyezhető el, így ha bonyolultabb szerkezetet kívánunk készíteni akkor konténer vezérlőbe való beágyazásra van szükségünk. Esetemben az egyes TItem-ek csupán egy SrcollViewer-ből áll, amelyek rendelkeznek egy Textblock-al, így nem volt konténervezérlőre szükség.

Végül egy Border elem maradt hátra az oldalamból. Ez a vezérlő eszközök elkülönítésére, kiemelésére szolgál. A elkülönítésre szánt controlok(at) segéllyel vehetjük körbe elérve, hogy az elemek kitűnjenek a többi közül. Csak egy gyermek eleme lehet ezért

ha több vezérlőt szeretnénk egy Border-ben elhelyezni akkor ismételten konténervezérlőt kell alkalmaznunk. Esetemben csak egy TextBlockot, használok, az utazás egy általános árának megjelenítésére.

3.3.2 A kereső felépítse

Kereső panelemet a MainPage.xaml fájlban implementáltam le. Egy Grid konténervezérlőben helyeztem el, tartalma pár TextBlock egy ComboBox, egy ListBox, egy Button és egy Slider. Program elindulásakor ebből mindössze egy „Állítsa össze katalógusát!" tartalmú Textblock, egy ComboBox és egy „Elkészít feliratú" Button látható. Gombra való kattintásra elindul a katalógus összeállítása a CombBox-ban és ListBox-ban beállított opcióknak megfelelően.

ComboBox

Általa elemek egy listáját jeleníthetjük meg, a listából alaphelyzetben egyetlen elem látható, mit a vezérlő SelectedIndex tulajdonságán keresztül állíthatunk be, egyébként nulla az alapértelmezetten kiválasztott listaelem. A ComboBox vezérlőre való kattintással láthatóvá válik az összes listaelem melyek közül kattintással egy kiválasztható. Tartalmát megadhatjuk ComboBoxItem-ek használatával vagy a vezérlő ItemSource tulajdonságán keresztül egy kollekcióhoz köthetjük.

Esetemben a ComboBox-t keresés személyre szabásában fogom felhasználni. Tartalmát a következőképpen adom meg:

```
<ComboBox x:Name="combobox" Height="28" Margin="25,61,0,0"
VerticalAlignment="Top" SelectedIndex="0" HorizontalAlignment="Left"
Width="138">
                <ComboBoxItem x:Name="all" Content="Összes Ajánlat!"/>
                <ComboBoxItem Content="Földrész"/>
                <ComboBoxItem Content="Ország"/>
                <ComboBoxItem Content="Ár"/>
                 </ComboBox>
```

„Összes ajánlat", „Földrész", „Ország", "Ár" alapján van lehetőség katalógusom összeállítására. A Combobox vezérlőmön code behind alól feliratkozok a SelectionChanged eseményre,

```
combobox.SelectionChanged += new
SelectionChangedEventHandler(combobox_SelectionChanged);
```

így mikor kiválasztok egy elemet a listából, aktivizálódik a `combobox_SelectionChanged` eseménykezelő metódus, mely egy if else szerkezetet implementál, minek megfelelő ága beállítja a Keres_Listbox ItemSource property-ét adatbázisban szereplő összes országot vagy kontinenst tartalmazó listára attól függően, hogy a „Földrész" vagy az „Ország" elemet választottuk-e ki a ComboBoxban. Ekkor a Keres_Listbox-ot láthatóvá teszem.

Listbox

A ComboBox-hoz nagyon hasonló vezérlő. Ugyanúgy elemeknek egy listáját jeleníti meg azzal a különbséggel, hogy akár több elem kiválasztására is lehetőségünk van és adott a méret függvényében egyszerre több elemet jeleníthet meg. Tartalmát megadhatjuk ListBoxItem-ek megadásával, melyeknek többnyire egyszerű elemek, de emellett összetett tartalmú Itemek használatára is lehetőség van konténervezérlő alkalmazásával. A másik lehetőség az ItemSource property-n keresztül megadott kollekció.

ListBox elemei az Utazas tábla soraiban megjelenő különböző országok vagy földrészek lesznek, a ComboBox-beli választás függvényében. Ha a ComboBox vezérlőkön az „Ár" elemet választjuk akkor két Slider és TextBlock-ok segítségével beállítható milyen ár-intervallumon belüli utazásokból szeretnénk válogatni.

Slider (Csúszka)

Egy beviteli vezérlő, segítségével numerikus értéket állíthatunk be fogója futófelületen való húzásával. Megadhatjuk a futófelület orientációját vízszintes függőleges értékekkel, továbbá beállíthatjuk a futófelület minimális, maximális értékét és egyéb finomhangoló tulajdonságokat. A panelben megjelenő egyik Slider-el a minket érdeklő utazások egy minimális árát, míg a másikkal az utazások maximális árát szabom meg. A változásokat egy-egy TextBlock segítségével jelenítem meg melyek Text tulajdonságát a megfelelő Slider-re kötöttem.

Az "Elkészít" gomb megnyomása után felépül a katalógus, ami egy beállított ár intervallumra vagy az adott országra vagy földrészre vonatkozó összes utazást fogja tartalmazni. Gombnyomáskor Keres_Button_Click eseménykezelőre kerül a vezérlés.

```
private void Keres_Button_Click(object sender, RoutedEventArgs e)
        {
            AdatforrasServiceClient client = new AdatforrasServiceClient
();
            loading.Opacity = 1;
            if (combobox.SelectedIndex == 0)
            {
                client.GetAllUtazasCompleted += new
EventHandler<GetAllUtazasCompletedEventArgs>(client_GetAllUtazasCompleted);
                client.GetAllUtazasAsync();
            }
            else if (combobox.SelectedIndex == 1) {
                client.GetKontintensUtazasCompleted += new
EventHandler<GetKontintensUtazasCompletedEventArgs>(client_GetKontintensUta
zasCompleted);

client.GetKontintensUtazasAsync(Keres_LisBox.SelectedItem.ToString());
            }...
```

Ahol példányosítok egy AdatforrasServiceClient proxy osztály, hogy elérjem lekérdezésemmel majd az adatbázist. If else szerkezet minden ágában egy aszinkron lekérdezés hívás látható és előtte ennek a lekérdezésnek a Completed (teljesített) eseményére való feliratkozása, ami majd a kívánt paraméterrel meghívja a MakeBook metódust.

3.3.3 Katalógus megjelenés

Silverlight-al először egy Microsoft által szervezett előadáson találkoztam, képek nézegetésére mutattak egy nagyszerű formát, mely a képek fotóalbumszerű megjelenítése volt, nagyon megtetszett és dolgozatomban ezért fel is használtam azzal a lényeges különbséggel, nem csupán képek jelennek meg az egyes oldalakon, hanem egyéb szöveges információk is. Mivel a könyv megjelenést nem Microsoft által készített vezérlő valósítja meg utánajárásom során kiderült, hogy létezik erre a célra előállított több nyílt forrású külső vezérlő is. Egy ilyen a SLMitsu által készített vezérlő is, amit felhasználtam szakdolgozatomban. Ez egyszerű módon az alkalmazás referenciái közé való felvétellel történt, majd a referenciában található UcBook kontrolt hozzáadtam a gyerekablakot reprezentáló ChildWindowKatalogus3.xaml fájlba. Funkcionális működéshez szükségünk van a vezérlő számára hozzáadni a tartalmazni kívánt lapokat, az Oldal UserControl osztály segítségével írom le a könyv egy lapját, tartalmát a MainPage osztály MakeBook metódusában állítom be ahol egyúttal a könyv tartalmat is beállítom. A metódus egyetlen paraméterrel rendelkezik, amely utazásokat tartalmazó ObservableCollection, ezt bejárva

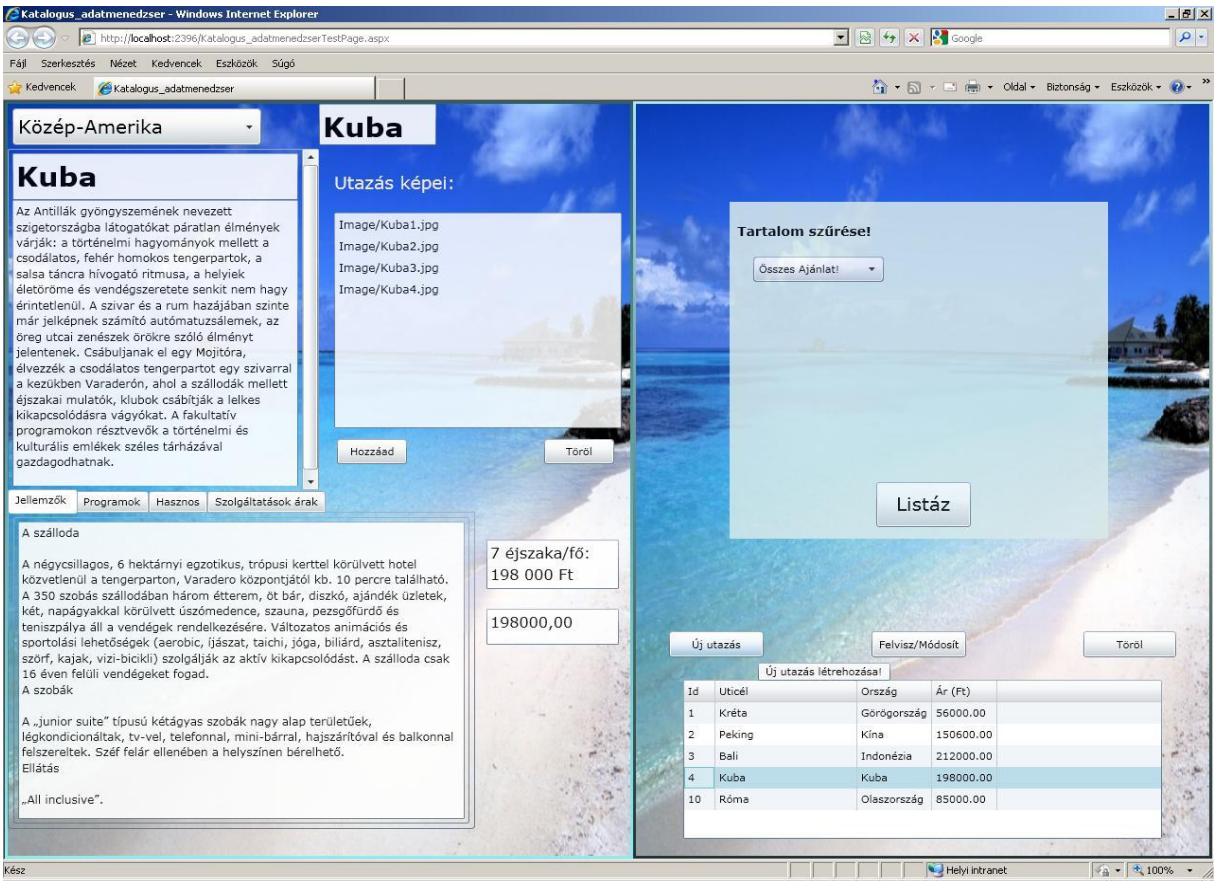
minden elemével egy új Oldal osztály hozunk létre melynek konstruktorát az adott utazás id-jével meghívjuk, erre azért van szükség, hogy az adott oldalhoz kinyerhessük a Media táblából az adott oldalhoz tartozó képeket. Az oldal TextBlockjainak Text része kötve van bizonyos objektum bizonyos mezőihez, de még nem adtuk meg azt az objektumot, ami rendelkezik a kötött mezőkkel ezért az Oldal DataContext tulajdonságával megadjuk az aktuális kollekció elemet, ennek köszönhetően az oldal vezérlői megkapják a számukra meghatározott információkat. Az egyes oldalakhoz hozzárendelem, az adott oldalon való balgomb lenyomása esetén a Click eseményre, mely eseménykezelő hajtódjon végre, az előrefele vagy hátrafele lapozás kezelése, ennek szerepe nem manuálisan lapozás esetén van. Ezen elhangzottakat reprezentálja a következő kódrészlet:

```
private void
MakeBook(System.Collections.ObjectModel.ObservableCollection<Utaza> Result)
        {
            for (int i = 0; i < Result.Count; i++)
            {

                oldaltemp.Add(new Oldal(Result[i].id));
                oldaltemp[i].DataContext = Result[i];
                gridtemp.Add(new Grid() { Margin = new Thickness(1),
Opacity = 1 });
                gridtemp[i].Children.Add(oldaltemp[i]);
                pages.Add(gridtemp[i]);
            }
            for (int n = 0; n < pages.Count; n++)
            {
                if (n % 2 == 0) pages[n].MouseLeftButtonDown += new
MouseButtonEventHandler(RightPage_MouseLeftButtonDown);
                else pages[n].MouseLeftButtonDown += new
MouseButtonEventHandler(LeftPage_MouseLeftButtonDown);
                 }

cwk.book.SetData(this);…
```

4. Katalógus adatmenedzser

Előző applikációm adatok megjelenítésére hivatott. Ezzel szemben az adatmenedzser a katalógus mögött meghúzódó adatbázis tábláit tudja módosítani Insert, Update, Delete utasításokkal. Jogok kezelését nem valósítottam meg, mivel ez egy belső felhasználásra szánt applikáció, ezáltal illetéktelen személyek nem férhetnek hozzá. Az alkalmazás egyetlen

oldalból áll mely, baloldala beviteli adatlapot jelenít meg a katalógus egy oldalának szerkezetéhez hasonlóan, míg jobb oldala tartalom navigációt és műveleti vezérlőket prezentál. Mivel a beviteli adatlapon egyetlen utazás adatainak módosítására van lehetőségünk ezért használtam egy táblázatot a navigáció megvalósítására, vagyis kiválasztható benne, hogy az adatbázis utazásokat tartalmazó táblájának mely sorát szeretnénk módosítani, esetleg törölni. A navigációs tábla tartalma az előző programban is használt keresőpanel segítségével meghatározott tartalomra szűkíthető. Az Új utazás gomb használatával új sor adható az utazásokat tartalmazó táblázathoz, ezt az új sort a Felvisz/Módosít gomb segítségével lehet véglegesíteni az adatbázisban. A Töröl gomb törli az adatlap tartalmat vagy az adatlap törlése mellett törli a kiválasztott sort az adatbázis Utazas táblájából ezáltal navigációs táblából is. A gombok rendelkeznek Tooltip szolgáltatással, mi által a felhasználók értesülnek az adott gomb mi célt szolgál, és egyébként az alkalmazást is könnyen átláthatóbbá teszi.



4.1 Adatbázis

Mivel a program, a Katalógus alkalmazás mögötti adatbázist kezeli Select, Insert Update, Delete műveleteken keresztül ezért ugyanazt az adatbázist használom, mint Katalógus. Ezen adatbázis részleteit dolgozatom 10. oldalán már feltüntettem, ezért nem kívánom még egyszer bemutatni. A részletekhez lapozzon vissza a 10. oldalra!

4.2 Alkalmazás és az adatbázis kapcsolata

Természetesen ezen program esetén is egy LINQ to SQL osztály segítségével képeztem le adatbázis típusaimat objektum orientált típusaik megfelelőivé. Az adatbázis elérésre szintúgy WCF kommunikációs technológiát használtam. Ezek általános leírását a dolgozatom 11-15. oldaláig részletem, további információért kérem lapozzon oda.

A szolgáltatás viszont kibővült az előzőekhez képest. A lekérdező műveletszerződések mellé került egy Insert, Update, Delete műveletek.

```
[OperationContract]
        public void AddUtazas(Utazas utazas_p)
        {
            adt.Utazas.InsertOnSubmit(utazas_p);
            adt.SubmitChanges();
             }
```

Látható a sorok kicsit sem hasonlítanak SQL nyelv Insert műveletének szintaktikájára, ennek ellenére az eredmény mégis ekvivalens lesz. Rekord felvitele az utazás táblába az adt AdatforrasDataContext objektumon keresztül történik. A DataContext-ben hivatkozk a táblára melybe sort akarunk beszúrni, erre a táblára meghívom az InsertOnSubmit metódust egy Utazas típusú paraméterrel, mely a táblához adja az új sort. Az adatbázisban mindaddig nem jelenik meg a változás, míg meg nem hívjuk a DataContext SubmitChanges() metódusát, ő végzi a változások mentését az adatbázison.

A módosító műveletszerződés hasonlóan egyszerű, a metódus a paraméterként kapott utazást lekérdezi az adatbázisból, majd az eredményül kapott utazás tulajdonságait a paraméter tulajdonságaival felülírja. Ezután meghívja a DataContext SubmitChages() metódusát az adatbázisban való változások mentésre.

Törlés a harmadik műveletszerződés mellyel kiegészült szolgáltatásom.

```
[OperationContract]
        public void DeleteUtazas(Utazas utazas_p)
        {
            var result = (from c in adt.Utazas
                          where c.id == utazas_p.id
                          select c).SingleOrDefault();
            if (result != null)
            {
                adt.Utazas.DeleteOnSubmit(result);
                adt.SubmitChanges();
             }}
```

A paraméterül megkapott utazást lekérdezem az adatbázisból és az eredmény a result váltózóba kerül. A SingleOrDefault() egy olyan metódus amely egy beállított default értékkel tér vissza (alkalmazásomban null) ha a lekérdezés eredménytelen volt vagy dob egy kivételt ha a lekérdezés eredménye egynél több sort tartalmaz. Használata esetemben igazából egy felesleges dolog így szerepe csak az ismertetésben van. Ugyanis az adatbázis és a lekérdezés jellegéből fakadóan nem beszélhetünk olyan esetről, hogy a lekérdezésnek akár nulla vagy egynél több sorral kellene visszatérni. Csak azért sem mert az Utazas tábla id egyezőségét vizsgálom a paraméter id-jével, ami egy egész típusú növekményes elsődleges kulcs. Feltételes szerkezetben megvizsgálom, hogy kaptam e valamiféle eredményt, erre szükség van, mert ha egy null értéket próbálok meg a DeleteOnSubmit segítségével kitörölni, akkor kivétel váltódik ki erre vonatkozóan. Meghívjuk a DataContext DeleteOnSubmit metódusát a keresés eredményével ami egy darab utazás, aztán a szokásos módon SubmitChanges() metódus.

4.3 Adatmenedzser UI felülete

Az alkalmazást a Katalógustól eltérően egy oldalon jelenítem meg, a felületen megjelent vezérlőket a teret kettéosztó két Border-ben helyeztem el. Ezáltal funkció szerint kettéosztva a felületet. A terület bal oldalán a katalógus egy oldalához tartozó váz jelenik meg. Az oldal fejlécében az ország vezérlő mellett megjelenik egy combobox_kontinensek vezérlőt, amely egy ComboBox vezérlő, tartalmát a Kontinensek tábla soraihoz kötöttem. Ezen vezérlő kiválasztott tartalma a katalógusomban információként nem jelenik meg, kitöltésére azonban mégis szükség van, ugyanis a katalógus összeállításánál megadtam mint az adatok szűrési feltétele. Az előző programban használt TextBlock-kokkal szemben, ezen alkalmazás WatermarkedTextBox-okat használ. Sajnos a Silverlight-ban ez a vezérlő alap eszközként

nincs jelen, így egy szabadon használható külső forrású WatermarkedTextBox került felhasználásra. Ez egy beviteli eszköz, szerkeszthető környezetet biztosít szöveg bevitelére. Az eszköz nagy előnye a Silverlight TextBox-ával szemben, hogy van egy Watermark (vízjel) tulajdonsága, melynek értéke megjelenik a vezérlő felületén, ha a vezérlő Text tulajdonsága nem rendelkezik értékkel és a vezérlő nincs fókuszban. A vízjel általában arról informál milyen beviteli jellegű szöveget vár a vezérlő. Az adatlapon megjelenő összes adatbeviteli mező WatermarkedTextBox. A továbbiakban leírom az oldal azon részeit, amelyek különböznek az előző program oldalától.

A Kep_Grid elrendezés vezérlőm tartalma megváltozott, tartalma egy Listbox melyben az éppen megjelenített utazás képeinek az adatbázisban megjelenő elérési útja található. Innen kitörölhetünk bejegyzéseket a Töröl gomb segítségével vagy újat adhatunk, a Hozzáad Gomb megnyomásával, minek hatására OpenFileDialog ablakon keresztül megadhatjuk az utazáshoz csatolandó képet. A megnyitott képek nem azonnal kerülnek be az adatbázisba, ugyanis nincs garancia, hogy az utazás mentésre kerül. Ha esetleg nem mentenénk, akkor fölöslegesen történt az adatbázisbeli művelet és persze a képfájl feltöltése is a szerverre. Ennek elkerülésére a képek csak egy string-eket tartalmazó kollekcióba kerülnek a további feldolgozás miatt. A mentést majd a Felvisz/Módosít gomb látja el.

A Border vezérlő kiegészült az előző programhoz képes egy újabb beviteli mezővel, tehát áll a megjelenített ár_tábla felvitelére alkalmas mezőből és az árszerinti keresést kiszolgáló ár felvitelére alkalmas mezőből. Azért választottam ezt az utat, mert az oldalakon az ár_tábla mezőben nem csupán egy árat szemléltető számokat szeretnék megjeleníteni, hanem mellette egy kiegészítő szöveget is. A másik mezőre, amit a másik program katalógus oldala nem jelenít meg, azért van szükség, mert az árszerinti keresést double típusú számok segítségével állítom össze és a lekérdezésben biztosítanom kell a hasonlító operátorok részére megfelelő típusegyezőségeket. Ezt a típusegyezőséget ugyan megtehetném az ártábla string-jéből is, de mivel ezt jóval körülményesebb lehetőségnek véltem, így az egyszerűbb megoldást választottam.

Az applikáció jobb felében kapott helyet az előző programból már ismert keresőpanel, továbbá egy DataGrid amely a keresőpanelen beállítottaknak megfelelően fog utazásokat megjeleníteni és további 3 műveleti gomb.

DataGrid (adattábla)

A vezérlő segítségével képesek vagyunk objektumok egy listáját táblázat formájában megjeleníteni az alkalmazásban. Lehetséges a megjelenített tartalom módosítása, törlése, esetleg új adatsor felvitele. ItemSource tulajdonságán keresztül kötjük a megjeleníteni kívánt objektumra. A kötött objektum tartalmának megfelelően generálja le az oszlopok számát a DataGrid, ezt a lehetőségünk van kikapcsolni az AutoGenerateColumns tulajdonság false értékre való állításával. Ekkor manuálisan kell megadnunk a megjelenítésre szánt oszlopokat:

```
<data:DataGrid x:Name="dg" Height="177" Margin="8,0,8,19"
VerticalAlignment="Bottom" AutoGenerateColumns="False"
SelectionMode="Single" SelectionChanged="dg_SelectionChanged"
d:LayoutOverrides="GridBox" >
                    <data:DataGrid.Columns>
                        <data:DataGridTextColumn Binding="{Binding id}"
Header="Id"/>
                        <data:DataGridTextColumn Binding="{Binding uticel}"
Header="Uticél"/>
                        <data:DataGridTextColumn Binding="{Binding orszag}"
Header="Ország"/>
                        <data:DataGridTextColumn Binding="{Binding ar}"
Header="Ár"/>

                         </data:DataGrid.Columns>…
```

Ekkor DataGridTextColumn objektumokon keresztül, egyesével meg kell adunk a megjelenítendő oszlop fejlécét, továbbá az ItemSource-nak megadott objektum mely jellemzőjét tartalmazza az oszlop. A táblázat egy sorának kiválasztásával a bevitelei adatlap mezőibe töltődnek a kiválasztott utazás megfelelő adatai. Ezt a funkciót az adattábla SelectionChanged eseményével érem el. Az esemény bekövetkezésekor, vagyis ha az adattáblában kiválasztok egy sort, a dg_SelectionChanged eseménykezelő veszi át a vezérlést. Hol megvizsgálom, hogy van-e új utazás felvitele folyamatban. Ha nincs és a kiválasztott sor nem nulla, akkor az adatlapon található összes WatermarkedTextBox szöveg tulajdonságát a kiválasztott sor megfelelő tulajdonságával teszem egyenlővé, továbbá lekérem az adott utazás képeit az adatbázisból, és az eredményt a kepek_Listbox forrásának állítom be. Mellette kezelem a képek forrását tartalmazó kollekciót.

Az első műveleti gomb az Új utazás, megnyomására a btn_uj_ut_Click eseménykezelő metódusra kerül a vezérlés.

```
private void btn_uj_ut_Click(object sender, RoutedEventArgs e)
        {
            AdatlapTorol();
```

```
uj_utazas = new Utazas();
btn_uj_ut.IsEnabled = false;
dg.SelectedIndex = -1;
uj_utazas_folyamatban = true;
 }
```

AdatlapTorol() eljárás hívásával törlöm az adatlap aktuális tartalmát a Text tulajdonságok üres string-re való állításával. Létrehozok egy az utazás számára egy megfelelő referenciát. Az Új utazás gombot inaktívvá állítom, kivédve hogy a felhasználó a gomb esetleges újbóli megnyomása esetén elvessze az adatlapra már felvitt adatokat. Aktiválása csak a Töröl vagy Felvisz/Módosít gomb megnyomására történik. Az adattábla kiválasztott sorát -1 értékre állítom minek a szerepe az utazás mentésénél lesz. Az uj_utazas_folyamatban logikai változó true értékre való állítására azért van szükség, mert előfordulhat olyan eset, amikor gomb inaktív és a felhasználó az adattábla valamely sorára vagy éppen a keresés panel Elkészít gombjára kattint. Ekkor hiba állna elő, mert ugye csak a Töröl és Felvisz/Módosít gomb aktiválja újra és ebben esetben az aktiválás nem történne meg vagyis a felhasználó a program újraindítása nélkül nem tudna új Utazást létrehozni. Ezen eset kezelésére vizsgálatot helyeztem el az adattáblám SelectionChanged eseménykezelőjében:

```
if (uj_utazas_folyamatban)
            {
                MessageBox.Show("Nem mentette az új utazást, mentse vagy
törölje az adatlapot!");
                dg.SelectionChanged -= dg_SelectionChanged;
                dg.SelectedIndex = -1;
                dg.SelectionChanged+=dg_SelectionChanged;
                return;
                 }
```

Amely megnézi uj_utazas_folyamatban változó segítségével van-e éppen mentésre váró új utazás, ha igen akkor egy szövegdoboz tájékoztatja a felhasználót a történtekről. A következő sorokra azért van szükség mert a mentés gomb a kiválasztott sor indexe alapján dönti el, hogy új bejegyzés mentéséről van szó vagy létező módosításáról. Eddig ez szám -1 volt, de a táblában sorválasztás történt, minek hatására a DataGrid SelectedIndex tulajdonsága megkapta a kiválasztott sor indexszámát, tehát ha nem állítanák vissza ennek értékét akkor a felhasználó a mentés, törlés gomb megnyomásával a kiválasztott sorral végezné el az adott műveletet nem pedig a mentendő új bejegyzésünkkel.

A Felvisz/Módosít gomb által új utazás vagy egy már létező utazás módosítása menthető az adatbázisba. Kattintás esemény, hatására meghívódik a btn_ment_Click eseménykezelő metódus.

```
private void btn_ment_Click(object sender, RoutedEventArgs e)
        {…
            try
            {
                uj_utazas.ar = Convert.ToDecimal(ar.Text);
            }
            catch (FormatException)
            {
                MessageBox.Show("Nem adott meg árat vagy érvénytelen
érték!");
                return;
            }…
            if (dg.SelectedIndex != -1)
                client.UpdateUtazasAsync(uj_utazas);
            else
                client.AddUtazasAsync(uj_utazas);

            btn_uj_ut.IsEnabled = true;
            AdatlapTorol();
            uj_utazas_folyamatban = false;
        }
```

Ekkor az utazás tábla egy sorát reprezentáló uj_utazas referencia tagváltozóit beállítom a beviteli adatlap megfelelő értékeivel. Ezek közül az ar és kontinens_id tagváltozókra nagyobb figyelmet fordítva, ugyanis ha az árat leíró WatermarkedTextBox nem megfelelő értéket tartalmaz, például kitöltetlen vagy esetleg jelen van benne számjegyen kívül valami egyéb karakter akkor ez a decimális számmá való konvertálásnál egy formátum kivételt FormatException fog dobni, valamint ha nem adunk értéket a kontinens_id-nek a combobox_kontinensek vezérlővel, akkor egy általános kivételt fog dobni Ezen lehetőség elkerülésére egy egyszerűsített validációs technikát alkalmaztam. Ha formátum kivétel váltódik ki a konvertálás során, akkor egy szövegdobozban elhelyezett információ segítségével tájékoztatom a felhasználót az ár mező értékének elhibázott megadására. Ha pedig egy általános kivétel az előzőhöz hasonló módon a felhasználó tájékoztatva lesz, hogy nem adott meg kontinenst a mentendő utazásnak.

Az adattábla kiválasztott indexének való vizsgálatával eldöntöm, hogy a mentés során új sor felviteléről van-e szó vagy sem, ennek alapján az uj_utazas paraméterrel aszinkron módon meghívom a szolgáltatásom AddUtazas vagy UpdateUtazas műveletszerződését.

A Töröl gomb töröl az adatbázisból egy a DataGrid által kiválasztott utazást, vagy töröli az adatlap WatermarkedTextBox-jainak szöveg értékét.

```
private void btn_torol_Click(object sender, RoutedEventArgs e)
        {
            if (dg.SelectedIndex != -1)
            {
                client.DeleteUtazasAsync(akt);
                AdatlapTorol();
            }
            else
            {
                AdatlapTorol();
            }
            btn_uj_ut.IsEnabled = true;
            uj_utazas_folyamatban = false;
             }}
```

Az előzőekhez hasonló módon eldöntöm a DataGrid kiválasztott sora alapján mit kell törölni adatbázisbejegyzést vagy csak beviteli adatlapon megjelenő adatokat. Adatbázis bejegyzés törlésénél csak az utazást törlő műveletet kell meghívni annak ellenére, hogy az aktuális utazásokhoz tartozó képek másik táblában van. Ennek oka, hogy a képek külső kulccsal kapcsolódnak az utazás id elsődleges kulcsához és a külsőkulcs definiálásakor megadtam a Cascade delete rule törlési szabályt. Ennek értelmében ha az utazás táblából törlődik egy sor akkor automatikusan a média táblából is törlődik az összes olyan rekord amely az utazás tábla törölt sorának külső kulcs által hivatkozott értékét tartalmazza. A program további helyes működése érdekében az uj_utazás _folyamatban logikai változó értékét hamisra állítom és az Új utazás gombot ismét aktívvá teszem.

ToolTip

Segítségével értesíthetjük a felhasználót az adott eszköz funkcióiról, használatával egy könnyen megérthető felhasználóbarát alkalmazást kaphatunk. Létrehozása a gomb definíciójában történt az alábbiak szerint:

```
<Button x:Name="btn_uj_ut" Height="29" HorizontalAlignment="Left"
Margin="35,0,0,221" VerticalAlignment="Bottom" Width="100" Content="Új
utazás"  Click="btn_uj_ut_Click" d:LayoutOverrides="GridBox" >
                    <ToolTipService.ToolTip >
                        <TextBlock Text="Új utazás létrehozása!" />
                    </ToolTipService.ToolTip>
                     </Button>
```

ToolTip-em esetemben egy TextBlock elem alkotja, de akár lehetőség van egy összetettebb forma kialakítására is. Ekkor az alkotó elemeket konténervezérlő gyermekeiként

kell elhelyezni, különben fordítási hibát kapunk, többszöri tulajdonság beállítással. Alkalmazásomban a ToolTip-ek csak egyetlen TextBlock-ból állnak.

5. Összegzés

Célom volt egy adatbázissal dolgozó RIA alkalmazás leimplementálásán keresztül bemutatni a Microsoft Silverlight keretrendszer nyújtotta eszközöket és a programhoz kapcsolódó technológiákat. Mindezt persze nem teljes részletességgel, de törekedtem egy átfogó körképet adni, főbb jellemzőiről. Bemutattam, hogy a Silverlight alkalmazások milyen módon jelennek meg a weboldalakon és levezettem, hogyan és milyen módon indul el ez a számítógépünkön. Ismertettem alkalmazásom, leírtam a készített nyilvántartó rendszerem funkcióját, felépítését és működését. Úgy gondolom minden lényeges információt megosztottam ahhoz, hogy az olvasó képessé váljon saját alkalmazást készíteni Silverlight-ban csupán a dolgozatom olvasatával.

Úgy gondolom az alkalmazás továbbfejlesztésével a felhasználói felületen való finomítással és az adott utazások biztonságos lefoglalását, megrendelését lehetővé funkció leimplementálásával egy vonzó felületet nyerhetünk az egyre nagyobb tömeget képviselő interneten vásárlók számára.

Köszönetnyilvánítás

Köszönöm Keczeli Csabának ki megadta a lehetőséget, hogy a CTS informatika Kft. – nél írjam szakdolgozatom, továbbá köszönettel tartozom Dvorák Péternek aki lektorálta és aki segítségemre volt a szakdolgozat megírásakor.

Köszönöm Mecsei Zoltánnak a Belső referensként elvégzett munkáját.

Irodalomjegyzék

1. Brad Dayley: **Silverlight 2 Bilbe**

   Wiley, 2008

2. Matthew MacDonald: **Silverlight 2 Visual Essentials**

   Apress, 2008

3. Joseph C. Rattz, Jr.: **Pro LINQ Language Integrated Query in C# 2008**

   Apress 2007

Hivatkozások

Silverlight

http://msdn.microsoft.com/en-us/bb187358.aspx

http://silverlight.net/

WCF

http://msportal.hu/blogs/turczy_attila/archive/2008/05/13/windows-communication-foundation-elm-233-leti-h-225-tt-233-r.aspx

LINQ

http://msportal.hu/blogs/zoltanarvai/archive/tags/LINQ/default.aspx

UCBOOK

http://blogs.msdn.com/mitsu/archive/2008/06/10/wpf-and-silverlight-bookcontrols-source-code-available.aspx