INFORMATIKAI TECHNOLÓGIÁK LABORATÓRIUM 2.

HALLGATÓI SEGÉDLET

WPF

1. Mérés célja

A mérés célja, hogy a hallgatók a WPF technológiát gyakorolják.

2. Szükséges ismeretek

A .NET platform alapismerete, C# nyelv, némi előismeret az adatkötés céljáról, WPF ismeretek.

3. Egyéb információk a méréshez

A mérésen egy számlázó programot kell készíteni, amely képes a már létrehozott megrendelésekből számlát készíteni és nyomtatni. Bár a mérés teljesen nulláról indul, egy számlát kirajzoló, előre megírt komponenst mégis használ, hogy a törvényeknek ténylegesen megfelelő formátumú számlát hozhassunk létre.

Beadni a jegyzőkönyvet kell, továbbá a kódot ZIP-elve.

Bónusz feladatok: A mérésen elérhető maximális pontszám 100, azonban a bónusz feladatokkal e fölé is lehet menni. Fontos, hogy a bónusz feladat pontszáma nincs összhangban a nehézségével, így csak azok fogjanak hozzá, akik amúgy készen vannak! További megkötés, hogy a laborvezető segítségét nem lehet kérni bónusz feladat megoldásához.

4. Mérési feladatok

4.1. Előkészítő lépések

a) Hozz létre egy projektet *Szamlazo* néven!
b) Ellenőrizd, hogy az adatbázis rendelkezésre áll-e, ha nem, akkor hozd létre a mellékelt script segítségével!
c) Ellenőrizd, hogy az adatbázisban vannak-e számlák és megrendelések!
d) A projektben hozz létre egy ShopDB.dbml fájt (LINQ to SQL)! A generált osztályhierarchiában az asszociációk neveit az angol többesszám miatt az „s” betűvel képezte. Ezeket javítsd ki a magyar helyesírás szerint. Például válaszd ki a Megrendeles és a Szamla közötti asszociációt, majd a Properties ablakon írjuk át a ChildProperties szekcióban a „Szamlas” nevet „Szamlak”-ra.

e) A szerver explorerbe vegyél fel egy új kapcsolatot, ami az adatbázisra mutat!
f) Jelöld ki az összes táblát, és húzd rá a ShopDB.dbml tervezői felületére!

4.2. Vezérlők elhelyezése (5p.)

a) A MainWindow-ban az alap Grid-hez adj hozzá két oszlopot! A jobb oszlop fix méretű legyen.
b) A bal oldali oszlopba tegyél bele egy DataGrid-et! Ez lesz a megrendelés lista.
c) A DataGrid legyen csak olvasható (IsReadOnly)
d) Jobb oldalra tegyél ki két gombot: Számla készítése, Számla nyomtatása feliratokkal!

4.3. Adatok felolvasása (5p.)

a) Hozz létre egy LoadDB metódust a MainWindow.xaml.cs-be! Ez fogja kilistázni a megrendeléseket.
b) Nyisd meg a ShopDB adatbázist a ShopDBDataContext segítésével! Ne felejts el using-ot használni!
c) LINQ segítésével kérdezd le az összes megrendelést!
d) A dataGrid.ItemsSource-nak add meg a lekérdezés eredményét!
e) Ideiglenesen adjuk meg, hogy hiába akar további adatokat lekérdezni a dataGrid, a kapcsolat már nem fog élni. Ezt a

```
db.DeferredLoadingEnabled = false;
```

utasítással tehetjük meg, ahol a db a ShopDBDataContext példánya. Ez később feleslegessé válik, de hibát nem fog okozni.
f) A konstruktorban hívd meg a LoadDB-t!

4.4. Adatok kiválasztása és konvertálása (10p.)

a) Hozzál létre egy MegrendelesModel nevű osztályt. Ebben tároljuk el minden egyes Megrendeles-hez azokat a property-ket, amiket meg akarunk jeleníteni a DataGriden.
b) Alakítsuk át a LoadDB függvényt, hogy minden egyes megrendeléshez egy MegrendelesModelt inicializáljon és ezek gyűjteményét kössük a DataGridhez.
c) A megrendelések bizonyos adatait akarjuk megjeleníteni. Mindegyikhez szükség van egy propertyre a modell osztályban, ezen kívül a LoadDB függvényben a modell osztály inicializálásakor gondoskodjunk arról, hogy az adatokat az alábbiak szerint a megfelelő formátumra konvertáljuk:
    - ID
    - Datum: formázhatjuk, hogy csak a dátumot mutassa:

```
megrendeles.Datum.Value.ToLongDateString( )
```

    - Telephely: mivel az adatbázisban három részre van szedve, állítsuk össze a nekünk tetsző formátumra. Például

```
string.Format( "{0} {1}, {2}", megrendeles.Telephely.IR,
megrendeles.Telephely.Varos, megrendeles.Telephely.Utca )
```

    - FizetesiMod:

```
megrendeles.FizetesMod.Mod
```

- Statusz:

```
megrendeles.Statusz.Nev
```

- Szamlazva

d) Továbbá írjunk ki némi információt a megrendelés tételeiről is (Megrendeles osztályon keresztül elérhető MegrendelesTetel gyűjtemény)
   - TetelSzam: a megrendelés tételeinek száma
   - Tetelek: A megrendelés tételei. Kezdetben ez string-ek listája legyen, amik a Termekek nevét tárolják.

e) Futtassuk az alkalmazást, és nézzük meg, hogy mit sikerült automatikusan kiírni.

4.5. DataGrid testre szabása (10p.)

Nem az igazi a DataGrid kinézete és nem is túl informatív a kiírt információ. Szabjuk testre DataTemplate használatával!

a) Definiálnunk kell az oszlopokat kézzel. Első körben hozzunk létre oszlopot minden adathoz és kössük hozzá a tulajdonsághoz. Például így:

```
<DataGridTextColumn Header="ID" Binding="{Binding ID}"/>
```

b) A tételekhez definiáljunk saját oszlopformátumot, amit a DataGridTemplateColumn oszloptípussal tehetünk meg. Ehhez meg kell adni a kinézetet is, ami általában így néz ki:

```
<DataGridTemplateColumn Header="Fejléc" Width="*">
    <DataGridTemplateColumn.CellTemplate>
        <DataTemplate>
            <TextBlock Text="{Binding IdeKellIrniATulajdonságot}"/>
        </DataTemplate>
    </DataGridTemplateColumn.CellTemplate>
</DataGridTemplateColumn>
```

c) Ez persze így még csak ugyanaz, mint egy sima DataGridTextColumn. Cseréljük le a belsejét egy olyan vezérlőre, ami tud megjeleníteni több elemet is. Legyen ez a ListBox, aminek az ItemsSource tulajdonságát beállítva lehet kiírni a tételeket.

d) Végül adjunk lehetőséget a felhasználónak, hogy alapban a sorok normál magasak legyenek és csak akkor mutassuk a tételeket, amikor kinyitja. Erre való az Expander vezérő. Egy példa az alkalmazására:

```
<Expander Header="{Binding MelyikTulajdonságLegyenKiírvaHaOsszeVanCsukva}">
    Vezérlő, ami kinyitott állapotbanLátszik adatkötéssel
</Expander>
```

e) „Számlázva” oszlop fordítása. A számlázva oszlopban jelenleg a „True”, vagy a „False” felirat jelenik meg a Szamlazva property értékétől függően. Hozzunk létre egy konvertert, aminek a segítségével ezek helyett az „Igen” és „Nem” szövegeket jeleníthetjük meg.

- A konverter osztály neve legyen BoolTranslatorConverter, az IValueConverter interfészt implementálja.
- Készítsünk egy statikus propertyt, aminek a neve Instance, a visszatérési értéke pedig egy BoolTranslatorConverter példány.
- A DataGrid megfelelő oszlopában a kötés létrehozásakor ne csak a property nevét definiáljuk, hanem adjuk meg a konvertert is:

```
Converter={x:Static  Szamlazo:BoolTranslatorConverter.Instance}
```

- Ennek a futtatásához még hiányzik a xaml fileból a Szamlazo névtér importálása, ezt elhelyezhetjük a Window objektumon nyitótagjébe:

```
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:Szamlazo="clr-namespace:Szamlazo"
```

4.6. Számla létrehozása (5p)

a) A DataGridtől az aktuálisan kiválasztott sorhoz tartozó MegrendelesModel típusú objektumot SelectedItem propertyvel kérdezhetjük le.
b) A gomb eseménykezelőjében ellenőrizzük, hogy van-e kiválasztott sor és nincs-e még számla létrehozva az adott megrendeléshez.
c) Hozzunk létre egy adatkapcsolatot (using használatával)
d) Generáljunk egy új azonosítót az új számlánknak:

```
var szlaid = ( from szamla in db.Szamlas select szamla.ID ).Max( ) + 1;
```

e) Hozzunk létre egy számlát a következő paraméterekkel:
   - ID = szlaid
   - MegrendeloNev = "Menza Márton"
   - MegrendeloIR = "1111"
   - MegrendeloVaros = "Budapest"
   - MegrendeloUtca = "Goldmann György tér 1."
   - FizetesiMod = "1"
   - KiallitasDatum = DateTime.Now
   - TeljesitesDatum = DateTime.Now
   - FizetesiHatarido = DateTime.Now
   - KiallitoID = 1
   - MegrendelesID = id

f) Szúrjuk be az adatbázisba: ”db.Szamlas.InsertOnSubmit(szamla);”

4.7. Számla tételek létrehozása (10p.)

a) Menjünk végig a megrendelés tételein, és minden tételhez hozzunk létre egy számla tételt is!
   - Másolnunk kell a Nev, Mennyiseg, NettoAr mezőket
   - Az AFAKulcs-ot nem tudjuk, állítsuk be 3-asra (20% legyen most)
   - A SzamlaID és a MegrendelesTetelID értelemszerűen kitölthető
b) Minden tételre hívjuk meg az InsertOnSubmit metódust!
c) Végül a függvény végén hajtsuk végre egyszerre az összes módosítást:

```
db.SubmitChanges( );
```

d) Majd frissítsük a felületet (pl hívjuk meg a LoadDB-t)!

4.8. Számla kirajzolása (10p.)

a) Adjuk hozzá a projekthez a mellékelt SzamlaOldal komponenst (SzamlaOldal.xaml és SzamlaOldal.xaml.cs)

b) A gomb eseménykezelőjében ellenőrizzük, hogy van-e kiválasztott sor és van-e hozzá már számla létrehozva.

a) Hozzunk létre egy SzamlaOldal példányt!

b) Nyissunk adatbázis kapcsolatot!

c) Kérdezzük le a számlát, amit nyomtatni kell (LINQ feltétel a szamla.MegrendelesID == id)!

d) A következő lépés a SzamlaOldal példány Adatok tulajdonságának kitöltése, de mivel a komponensünk nem teljesen azonos az adatbázissal, kénytelenek vagyunk kézzel konvertálni. Próbáljuk meg az összes lehetséges oszlopot átadni, és csak ott improvizálni, ahol nincs adatunk:

- Peldany = "1. (eredeti)"
- Szamlaszam = string.Format("A-{0}/{1:0000}", DateTime.Now.Year, DateTime.Now.Millisecond)

4.9. Számla tételek kirajzolása (10p.)

a) A Tetelek tulajdonság kitöltéséhez konvertálni kell a számla tételeit a megfelelő formátumra. Tudjuk a Nev, Mennyiseg, Egysegar mezőket, de valamit ki kell találjunk a Cikkszam és AFASzazalekhoz:

- Cikkszam = "11.11.11"
- AFASzazalek = 25

b) Jelenítsük meg a számlát egy új ablakban (Vie wbox használatával pont úgy látjuk, ahogy nyomtatni lehetne):

```
new Window {
  Content = new Viewbox { Child = szamlaOldal }, Width = 600 }.Show( );
```

4.10. Számlatételek részletezése (10p)

A MegrendelesModel osztályban eddig string-ek gyűjteményeként tároltunk a megrendeléshez kapcsolódó tételek nevét. Hozzunk létre egy külön osztályt (MegrendelesTetelModel) a tételek számára, ebben tároljuk a nevet (Nev) és a nettó árat (NettoAr).

a) Alakítsuk át a MegrendelesModel-t úgy, hogy a Tetelek propertyben MegrendelesModel-ek gyűjteményét tároljuk.

b) Módosítsuk a LoadDB függvényt, hogy megfelelően inicializálja a módosított megrendelés modelleket.

f) Módosítsuk a ListBox sablonját, hogy a következő formátumban jelenítse meg az adatokat: „név (nettó ár)”. ListBox esetén is DataTemplate segítségével definiáljuk az egyes elemek kirajzolásához szükséges sablont!

```
<ListBox>
```

```
      <ListBox.ItemTemplate>
            <DataTemplate>
                  …
             </DataTemplate>
       </ListBox.ItemTemplate>
</ListBox>
```

g) Figyeljünk arra, hogy a DataTemplate-en belül csak egy objektum állhat, ezért ha ide több vezérlőt szeretnénk elhelyezni, akkor először valamilyen konténer típusú objektumra (pl. StackPanel, DockPanel, Grid) lesz szükség.

4.11. Gombok engedélyezése (5p)

Gondoskodjunk arról, hogy a gombok csak akkor legyenek engedélyezettek, ha az adott műveletet valóban végre is tudjuk hajtani.

- Kezdetben mindkét gomb IsEndbaled propertyje legyen false.
- Ha megváltozik a kijelölés a DataGriden, akkor frissítsük a gombok állapotát. (Segítség: használjuk a DataGrid SelectionChanged eseményét!)

4.12. Megrendelés adatainak beolvasása (20p)

a) A számla létrehozásánál a megrendelő nevére és címére alapértelmezett adatokat használtunk eddig. Alakítsuk át a programot úgy, hogy a számlakészítés gomb megnyomásakor megjelenjen egy ablak, amiben megadhatjuk a megrendelő adatait (név, irányítószám, város, utca).

b) A számla létrehozásánál meg kell adni egy kiállító azonosítót is, eddig az 1-es azonosítót használtuk. A lehetséges kiállítókat a SzamlaKiallito táblában tároljuk. A számlakészítésnél megjelenő ablakban jelenítsük meg valamilyen módon a lehetséges számlakiállítókat, hogy a felhasználó tudjon közülük választani. A megjelenítéshez használhatunk például DataGridet, ComboBoxot, ListBoxot. Nem kötelező táblázatosan a kitállítók minden adatát megmutatni.

c) Végezetül a lehetséges fizetési módokat is adatbázisból olvassuk ki (FizetesMod tábla), jelenítsük meg és a kiválasztott szerint állítsuk ki a számlát. A fizetésmódok tárolnak egy határidőt is, ez kell legyen a teljesítés dátuma és a fizetési határidő közötti napok száma, erre is figyeljünk a dátumok megadásakor.

4.13. Bónusz feladat (5p) – képek megjelenítése

a) Vegyünk fel a MegrendelesTetelModel osztályba egy Kep nevű propertyt, aminek a típusa Binary (System.Data.Linq) legyen. Ebben fogjuk eltárolni az adott tételhez tartozó képet, ha az bent van az adatbázisban.

b) Módosítsuk a ListBox-ban található DataTemplate-et, hogy ToolTip-ben megjelenjen az adott termék képe. A képet egy Image vezérlővel jelenítsük meg. Ennek a Source propertyjével tudjuk megadni a kép forrását, ez azonban egy ImageSource típusú objektumot vár. Készítsünk egy konvertert, ami a korábban tárolt Binary típusú adatot átalakítja, majd használjuk a konvertert a Binding objektumban.

Segítség:

- Az alábbi kódrészlet megmutatja, hogyan tudunk egy byte tömböt ImageSource típusúra alakítani. A Binary típus valójában egy byte tömböt tárol, amit a ToArray() metódussal kérdezhetünk le.

```
MemoryStream stream = new MemoryStream(bytearray);
BitmapImage image = new BitmapImage();
image.BeginInit();
image.StreamSource = stream;
image.EndInit();
```

A másik lehetőség a System.Windows.Media.ImageSourceConverter osztály használata, aminek a ConvertFrom függvényével tudunk egy byte tömböt képpé alakítani.

- A legtöbb vezérlőnek van ToolTip propertyje, amibe tetszőleges vezérlőt elhelyezhetünk. Egy TextBlock esetén például így lehet definiálni:

```
<TextBlock>
      <TextBlock.ToolTip>
            tooltip
      </TextBlock.ToolTip>
</TextBlock
```

4.14. Bónusz feladat (5p.) – összeg kiírása betűvel

a) A számla alján az összeg betűvel kiírása nincs implementálva. (SzamlaOldal.xaml.cs-ben a BruttoSzovel tulajdonság). Implementáljuk!

- Ne feledjük, a helyesírási szabály az, hogy kétezerig egybe kell írni a szavakat, kétezer fölött ezres csoportonként kötőjellel.
- Például
  - 5432: ötezer-négyszázharminckettő
  - 1999: ezerkilencszázkilencvenkilenc
  - 1234567: egymillió-kétszázharmincnégyezer-ötszázhatvanhét

4.15. Bónusz feladat (10p.) – expander animálása

Animáljuk az Expander kinyílását és csukódását, hogy szebb eredményt kapjunk.

Segítség: az egyik lehetséges megoldás, hogy feliratkozunk az expander Expanded és Collapsed eseményeire. Ezekben létrehozunk egy-egy DoubleAnimation példányt, amivel majd az expander magasságát tudjuk módosítani. A DoubleAnimation kezdő és végértékének megadásakor szükség lehet az egyes vezérlők aktuális magasságának lekérdezésére (ActualHeight property), illetve a ListBox magasságának kiszámítására. Ez utóbbinál feltehetjük, hogy minden elemét egy 18 pixel magas téglalapban jeleníti meg. Egy vezérlőn egy animációt a BeginAnimation függvény segítségével indíthatunk el.

4.16. Bónusz feladat (15p) – lapozás a számlán

Alakítsuk át a SzamlaOldal.xaml-ben található vezérlőt úgy, hogy túl sok tétel esetén több oldalra bontja lapozhatóan a tételeket!

5. Ellenőrző kérdések

5.1. Mi az a dependency property?

5.2. Mi az attached property?

5.3. Mi a különbség a logical tree es a visual tree között?

5.4. Milyen elrendezést definiál az a grid, ami két sort definiál a következő Height propertykkel: Height="" illetve Height="2*"*

5.5. Mire szolgál a binding osztály?

5.6. Milyen értékeket tartalmazhat a Binding Mode property-je

5.7. Milyen interface-t kell megvalósítania egy converter-nek

5.8. Mi a különbség egy List<T> és egy ObservableCollection<T> típusú property között változásértesítés szempontjából.