4. LABOR - XAML 1

Segédlet és jegyzőkönyv sablon a Szoftverfejlesztés laboratórium 2 c. tárgyhoz

Tibor Tóth; Kővári Bence; Szücs Cintia Lia

2020

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

1117 Budapest, Magyar tudósok körútja 2.
Q. ép. B. szárny 207. • www.aut.bme.hu
Telefon: 463-2870 • Fax: 463-2871

Bevezetés

Célkitűzés

A labor célja, hogy a hallgatók gyakorolják a Kliens oldali technológiák c. tárgyban tanult XAML ismereteket.

Előfeltételek

A labor elvégzéséhez szükséges eszközök:

- Microsoft Visual Studio 2019
  - Universal Windows Platform v19041 SDK

Amit érdemes átnézned

- Universal Windows Platform jellemzői
- Kliens oldali technológiák jegyzet - XAML technológiák
- Kliens oldali technológiák labor – XAML laborok anyaga

A labor értékelése

A labor teljesen önálló munka végzésére szolgál. A teljes munkát a laborvezető a labor végén értékeli. Az értékeléskor a laborvezető kérdéseket tehet fel az elkészült alkalmazással kapcsolatban, amik befolyásolhatják az érdemjegyet. Az értékelést a hallgató a labor végeztével, de legkésőbb a laborvezető által a labor megkezdésekor jelzett időpontban jelzi a laborvezetőnek. Az ellenőrzéshez segítséget nyújt a labor végén összeállított ellenőrző lista.

Távolléti formában: a megadott határidőig kell mindenkinek feltölteni a labort, a határidő lejárta után értékelik a laborvezetők a leadott munkákat.

A labor végén a tárgy moodle oldalára (https://edu.vik.bme.hu/) feltöltendő az elkészült forráskód ZIP formátumban. A ZIP-ből törlendők a fordítási artifaktok (bin, obj mappák)!

Projekt létrehozása

Egy Universal Windows Platformon készült alkalmazást készítünk el a labor során, ami teendőket fog tudni kezelni. Ezen az alkalmon csak a felületet rakjuk össze minimális üzleti logikával, a második alkalmon pedig az MVVM minta használatával strukturáljuk az alkalmazásunkat, és az üzleti logika is közelíteni fog a valós felhasználáshoz.

Hozzunk létre egy új C# / Windows Universal / Blank App projektet **<neptunkód>** néven.

A Target és a Minimum verzió is legyen most version 2004 (19041). Amennyiben ez a laborgépen nem áll rendelkezésre, válasszuk a legfrissebb elérhető verziót is.

Próbáljuk ki, hogy fordul-e és fut-e az üres projekt!

Modell kialakítása

Hozzuk létre az adatmodellünket a projekten belül egy új `Models` mappába. A `TodoItem` osztály fogja tartalmazni a teendők adatait:

```
public class TodoItem
{
    public int Id { get; set; }
    public bool IsDone { get; set; }
    public string Title { get; set; }
    public string Description { get; set; }
    public DateTimeOffset Deadline { get; set; }
    public Priority Priority { get; set; }
}
```

A prioritáshoz egy felsorolt típust hozzunk létre egy külön fájlban:

```
public enum Priority
{
    High,
    Normal,
    Low
}
```

Főoldal

Adatok

A `MainPage` nézet fogja a teendők listáját megjeleníteni. Most memóriában lévő tesztadatokkal használjunk, amit a `MainPage.xaml.cs`-ben hozzunk létre:

```
public sealed partial class MainPage : Page
{
    public List<TodoItem> Todos { get; set; } = new List<TodoItem>()
    {
        new TodoItem()
        {
            Id = 1,
            Title = "Tejet venni",
            Description = "Ha van tojás, hozz tizet!",
            Priority = Priority.Normal,
            IsDone = true,
            Deadline = DateTimeOffset.Now + TimeSpan.FromDays(1)
        },
        new TodoItem()
        {
            Id = 2,
            Title = "Megírni a szakdolgozatot",
            Description = "Minimum 40 oldal, szépen kitöltve screenshotokkal",
            Priority = Priority.High,
            IsDone = false,
            Deadline = new DateTime(2017, 12, 08, 12, 00, 00, 00)
        }
    };
```

```
    public MainPage()
    {
        this.InitializeComponent();

        DataContext = this;
    }
}
```

A `Todos` tulajdonság fogja tartalmazni a listát, amit kikötünk a felületen a listában. Ez a lista `TodoItem` objektumokat tartalmaz.

Mivel adatkötni (`Binding`) szeretnénk a felületen, meg kell határozzuk az adatkötés alapértelmezett forrását, ami a `DataContext` tulajdonság értéke. Most nem alkalmazunk MVVM mintát (köv. labor), így most fapados megoldásként legyen ez az érték maga az oldal (`this`).

Nézet

A `MainPage.xaml` nézet, gyökér `Grid` vezérlőjébe készítsük el a listát. Egyelőre a listaelemek tartalmazzák egymás alatt a teendő címét és határidejét.

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <ListView ItemsSource="{Binding Todos}">
        <ListView.ItemTemplate>
            <DataTemplate>
                <StackPanel>
                    <TextBlock Text="{Binding Title}" FontWeight="SemiBold"/>
                    <TextBlock Text="{Binding Deadline}"/>
                </StackPanel>
            </DataTemplate>
        </ListView.ItemTemplate>
    </ListView>
</Grid>
```

A listák elemeit az `ItemsSource` tulajdonságban adjuk meg adatkötés (`Binding` markup extenisonnel): a `DataContext` (maga az oldal) `Todos` tulajdonságához kötjük azt. A lista elemek kinézetét az `ItemTemplate` tulajdonságban specifikálhatjuk egy `DataTemplate` segítségével, ahol az adatkötés `DataContext`-e már az adott listaelem (`TodoItem`) lesz, így annak a tulajdonságaihoz kötünk.

Próbáljuk ki!

Próbáljuk ki szimulátorban is a futtatást!

A későbbiekben minden feladatot tetszőlegesen tesztelhetünk szimulátorban, vagy asztali környezetben.

KONVERTER HASZNÁLATA

Jelenítsünk meg a listaelemekben egy pipával azt, hogy az adott elem készen van-e? (`IsDone`) Ehhez használjunk az adatkötés során konvertert. A feladatunk egy `SymbolIcon` `Visibility` tulajdonságára kikötni az elem `IsDone` tulajdonságát úgy, hogy az igaz-hamis értéket egy konverter transzformálja láthatóság típussá.

Hozzunk létre egy új mappát `Converters` néven majd bele egy új osztályt `BoolToVisibilityConverter` néven, ami megvalósítja az `IValueConverter` interfészt az adatkötés számára:

```
public class BoolToVisibilityConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, string language)
    {
        return (bool)value ? Visibility.Visible : Visibility.Collapsed;
    }

    public object ConvertBack(object value, Type targetType, object parameter, string language)
    {
        throw new NotImplementedException();
    }
}
```

Konvertert tipikusan XAML-ben statikus erőforrásként használunk. Ehhez vegyük fel a `MainPage` kódjába az oldal erőforrásai közzé egy példányt a konverterből. (ne felejtsük el az xml namespace-t sem)

```
xmlns:c="using:TodoXaml.Converters"

<Page.Resources>
    <c:BoolToVisibilityConverter x:Key="BoolToVisibilityConverter"/>
</Page.Resources>
```

Innentől kezdve az `x:Key`-ben megadott kulccsal tudunk hivatkozni az erőforrásra a `StaticResource` markup extensionnel az oldalon belül. Ha alkalmazás szintű erőforrást szeretnénk létrehozni, akkor azt az `App.xaml` fájlban kell megtegyük.

```
<DataTemplate>
    <StackPanel>
        <Grid>
            <TextBlock Text="{Binding Title}" FontWeight="SemiBold"/>
            <SymbolIcon HorizontalAlignment="Right" Symbol="Accept"
    Visibility="{Binding IsDone, Converter={StaticResource BoolToVisibilityConverter}}"/>
        </Grid>
        <TextBlock Text="{Binding Deadline}"/>
    </StackPanel>
</DataTemplate>
```

Próbáljuk ki!

Részletes oldal

Létrehozás

Hozzunk létre egy új oldalt a teendő adatainak szerkesztésére. Viszont előbb helyezzük át az eddigi `MainPage` nézetünket egy új `Views` mappába. A `Views` mappába hozzuk létre a `TodoDetailsPage` oldalt (Add / New Item / Blank Page).

Navigáció

A főoldalra rakjunk ki egy gombot, ami az új elem létrehozását végiz úgy, hogy elnavigál az üres részletes oldalra.

A gombot, most ne `Button` vezérlővel készítsük el, hanem használjuk a `CommandBar` vezérlőt, amibe még az oldal címét helyezzük el pluszban. Ehhez a főoldalon a gyökér `Grid`-et két sorra kell bontanunk, ahol az első sorban lesz a `CommandBar`, a másodikban pedig a lista:

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>

    <CommandBar Grid.Row="0" DefaultLabelPosition="Right" VerticalContentAlignment="Center">
        <AppBarButton Icon="Add" Label="Hozzáadás"/>
        <CommandBar.Content>
            <TextBlock Text="Teendők" Style="{ThemeResource TitleTextBlockStyle}" Margin="12,0,0,0"/>
        </CommandBar.Content>
    </CommandBar>
    <ListView Grid.Row="1" ItemsSource="{Binding Todos}">
...
```

A `CommandBar` többek között `AppBarButton` vezérlőket képes kezelni, de `Content`-ként bármilyen tartalom megadható, ami a cím esetünkben. Ha kész vagyunk vele, iratkozzunk fel az `AppBarButton` `Click` eseményére, ahol navigáljunk át az oldal által hivatkozott szülő `Frame` objektummal a részletes oldalunkra. A navigáció során paraméterként most `null`-t adjunk át.

```
<AppBarButton Icon="Add" Label="Hozzáadás" Click="AddButton_Click"/>

private void AddButton_Click(object sender, RoutedEventArgs e)
{
    Frame.Navigate(typeof(TodoDetailsPage), null);
}
```

A részletes oldalra is vegyünk fel egy `CommandBar`-t, amibe az oldal címét „Részletek” írjuk. A `CommandBar` stílusát most még csúnyán copy-paste-eljük át a főoldalról és írjuk át az oldal címét, majd **próbáljuk ki az alkalmazást!**

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <CommandBar DefaultLabelPosition="Right" VerticalContentAlignment="Center">
        <CommandBar.Content>
            <TextBlock Text="Részletek" Style="{ThemeResource TitleTextBlockStyle}" Margin="12,0,0,0"/>
        </CommandBar.Content>
    </CommandBar>
</Grid>
```

Vissza navigáció

Mit tapasztalunk? Nem tudunk visszamenni, mert nincs vissza gomb az ablakban. Az UWP alkalmazásoknak beépítetten van vissza gomb támogatásuk (telefon hardveres vissza gombja, tablet módban start menü melletti vissza gomb, ablakok bal felső sarkában). Hogy ezt ki tudjuk használni össze kell kötnünk ennek a működését a `Frame` navigációs működésével. A `Frame`-et az `App.xaml.cs OnLaunched` metódusban hozzuk létre, itt iratkozzunk fel ott `Navigated` eseményére, ahol kirakjuk az ablak címsorába a vissza gombot, a `SystemNavigationManager` osztály segítségével. Illetve iratkozzunk fel a `SystemNavigationManager BackRequested` eseményére, ahol elsütjük a vissza navigációt a `Frame`-en:

```
rootFrame = new Frame();

rootFrame.Navigated += (sender, args) =>
{
    SystemNavigationManager.GetForCurrentView().AppViewBackButtonVisibility =
       rootFrame.CanGoBack ? AppViewBackButtonVisibility.Visible : AppViewBackButtonVisibility.Collapsed;
};

SystemNavigationManager.GetForCurrentView().BackRequested += (sender, args) =>
{
    if (rootFrame.CanGoBack)
    {
        rootFrame.GoBack();
    }
};
```

Próbáljuk ki!

Ezzel elkészültünk az alkalmazás lényegi részével. A labor ezen pontjáig eljutva **közepes** érdemjegy kapható.

Önálló feladatok

1. Új bejegyzés felvétele - Űrlap

Készítsük el a részletes oldalt. Az alábbi vezérlőket használjuk az űrlap elkészítéséhez:

- `IsDone` ➔ `CheckBox`
- `Title` ➔ `TextBox`
- `Description` ➔ `TextBox`
- `Priority` ➔ `ComboBox`
- `Deadline` ➔ `DatePicker`

Használjunk mindenhol adatkötést a főoldalhoz hasonló módon.

Tipp: A ComboBox elemei meghatározásához használhatjuk a következő kódrészletet:
```
Enum.GetValues(typeof(Priority)).Cast<Priority>();
```

---

2. Új bejegyzés felvétele - Mentés

Vegyünk fel egy mentés műveletet a részletes oldalra, ami jelenleg az új elem mentését tudja lekezelni.

Most csúnya megoldásként tegyük statikussá a `MainPage`-ben lévő `Todos` listát, és ide dolgozzon a mentés művelet.

Mentés után navigáljunk vissza.

3. Új bejegyzés felvétele - MainPage példány újrafelhasználása

Állítsuk át a `MainPage` objektum gyorsítótárazhatóságát a konstruktorban:

```
public MainPage()
{
    this.InitializeComponent();
    NavigationCacheMode = NavigationCacheMode.Required;
    DataContext = this;
}
```

Ilyenkor a vissza navigáció során nem példányosít új `Page` objektumot a rendszer, hanem az előző példányt használja fel újból.

Teszteljük a 2, feladatban elkészített viselkedést! Mit tapasztalsz?

Javítsd ki az alkalmazást úgy, hogy a cache viselkedését Required értéken hagyod!

4. Teendők lista – listaelem navigáció

Navigálj a listaelemre történő kattintás hatására a részletes oldalra. Ne úgy old meg, hogy gomb vezérlőt helyezel el a listaelem sablonban.

A **mentés logikáját** nem kell módosítani.

Tipp: Használjuk az OnNavigatedTo metódust a navigálás során.

---

5. TEENDŐK LISTA - FELTÉTELES SZÍNEZÉS

Színezze ki a teendők hátterét különböző színűre a prioritásuk alapján, ehhez használjon adatkötést és konvertert. Extra jó megoldás: C# kódba égetett értékek helyett legyen lehetőség megadni XAML-ből a színeket.

Tipp: XAML-ből egy objektum tulajdonságait, (eseménykezelőit) állíthatjuk be.

---

6. COMMANDBAR - KÓDDUPLIKÁCIÓ

Szüntesd meg a két CommandBar-ban található felesleges kódduplikációt! (`DefaultLabelPosition`, `VerticalContentAlignment`, `Content` megjelenítése)

Tipp: Globális Style erőforrás létrehozása + Content helyett ContentTemplate megadása amiben, ha egyszerű {Binding}-ot hívunk, akkor a Content tulajdonság tartalma kerül kötésre.

Az önálló feladatok megoldása **egyenként +0.5-et (összesen legfeljebb 2-t)** ad az érdemjegyhez.