Eötvös Loránd Tudományegyetem
Informatikai Kar

Eseményvezérelt alkalmazások

---

9. előadás

Bevezetés az Avalonia UI keretrendszerbe

Dr. Cserép Máté
mcserep@inf.elte.hu
https://mcserep.web.elte.hu

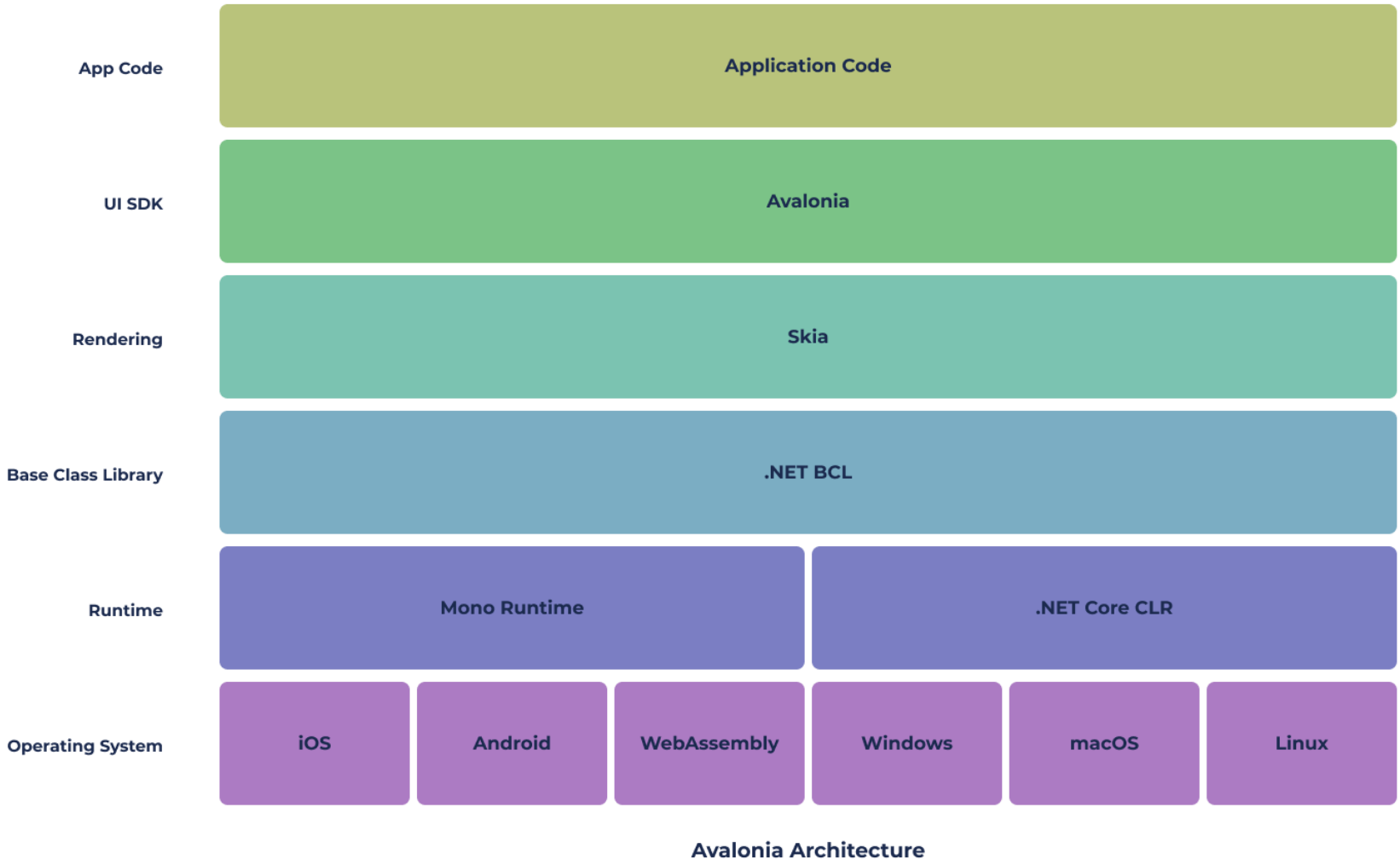
Avalonia UI alapismeretek

Tulajdonságai

- Az *Avalonia UI* egy a .NET környezetre épülő, multi-platform keretrendszer eseményvezérelt grafikus alkalmazások fejlesztésére
  - Windows, Linux, macOS, Android, iOS platformok támogatása, továbbá WebAssembly támogatás
  - alapja a .NET keretrendszer, amely cross-platform fejlesztést tesz lehetővé a támogatott platformok között
  - támogatja a fejlesztést MV architektúrában is, de kifejezetten az MVVM architektúra használata javasolt
  - lehetőséget ad a felület deklaratív leírására (*XAML*)
  - nézetmodellje kompatibilis a WPF és MAUI alkalmazások nézetmodell rétegével (megfelelő tervezés esetén)
  - opcionálisan lehetővé teszi a *reaktív programozás*t

Avalonia UI alapismeretek

Architektúra



Avalonia Architecture

Avalonia UI alapismeretek

Telepítés

- Az Avalonia NuGet csomagok telepítésén keresztül könnyedén egy létező projekthez adható, azonban a kezdeti fejlesztéskor hasznos lehet az előre készített sablon projektek (*template*-ek) használata.
  - Telepítés parancssorból:

```
dotnet new install Avalonia.Templates
```

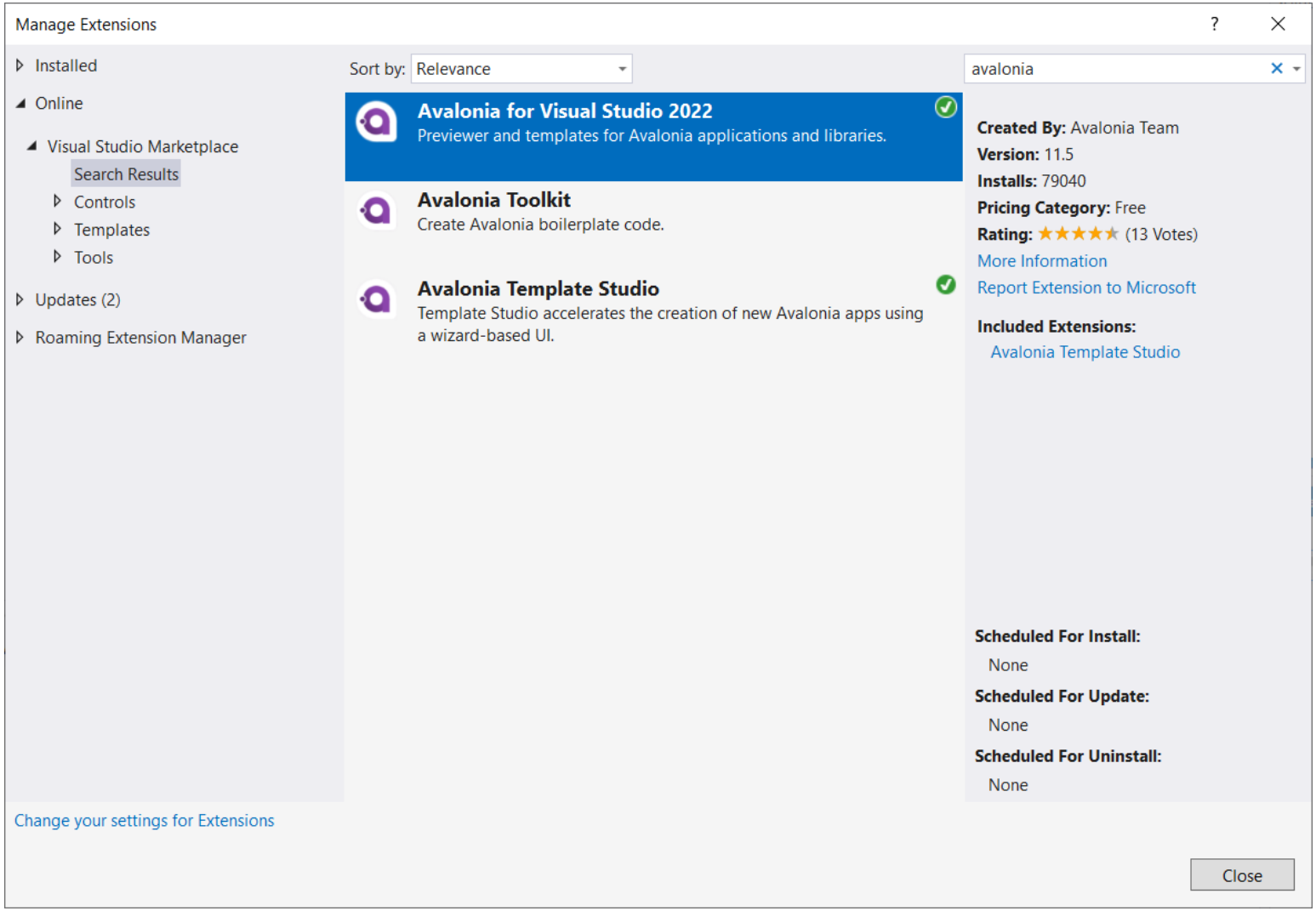
  - Majd új MVVM alkalmazás létrehozása:

```
dotnet new avalonia.mvvm -o MyApp -n MyApp
```

- Visual Studio használata esetén célszerű az *Avalonia for Visual Studio 2022* telepítése, így nem cask a sablonok, hanem a grafikus felület tervezőt is használhatjuk.
  - JetBrains Riderhez is elérhető hasonló támogatás.

Avalonia UI alapismeretek

Telepítés



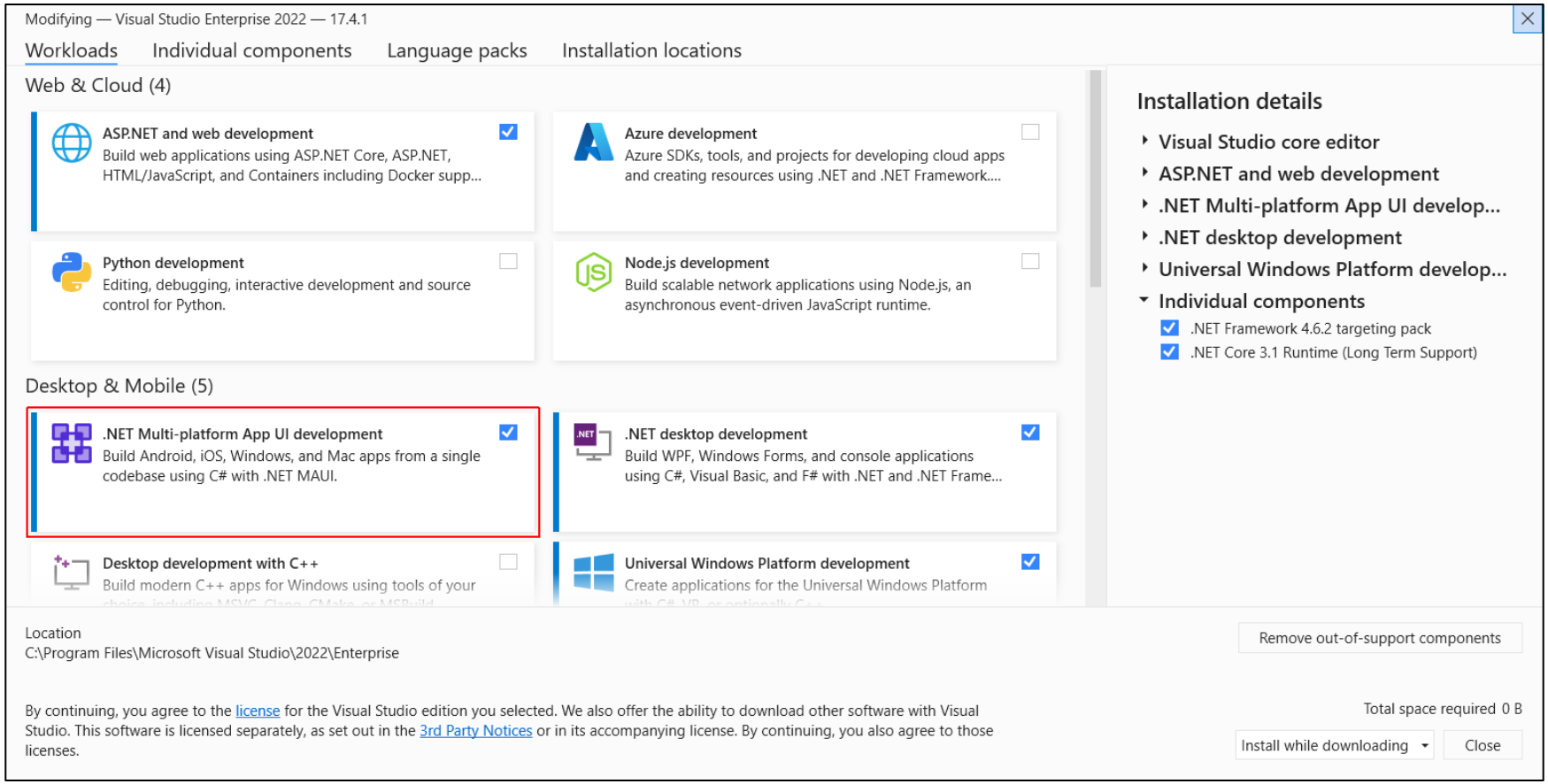
Avalonia UI alapismeretek

Telepítés

- Android platformra fejlesztés támogatásához telepítenünk szükséges az Android SDK-t (opcionálisan emulátort), valamint a Java SDK-t
  - A legegyszerűbb megoldás, ha a *MAUI workload*-ot telepítjük a Visual Studio telepítőjének újrafuttatásával
  - Linux alatt a **`dotnet workload install android`** parancs kiadásával telepíthető a minimálisan szükséges *workload*, majd Android SDK és Java SDK külön telepítése szükséges még
- WebAssembly fordítás támogatásához a *WebAssembly Build Tools* csomag telepítése szükséges az *ASP.NET and web Development workload* nem kötelező csomagjai közül

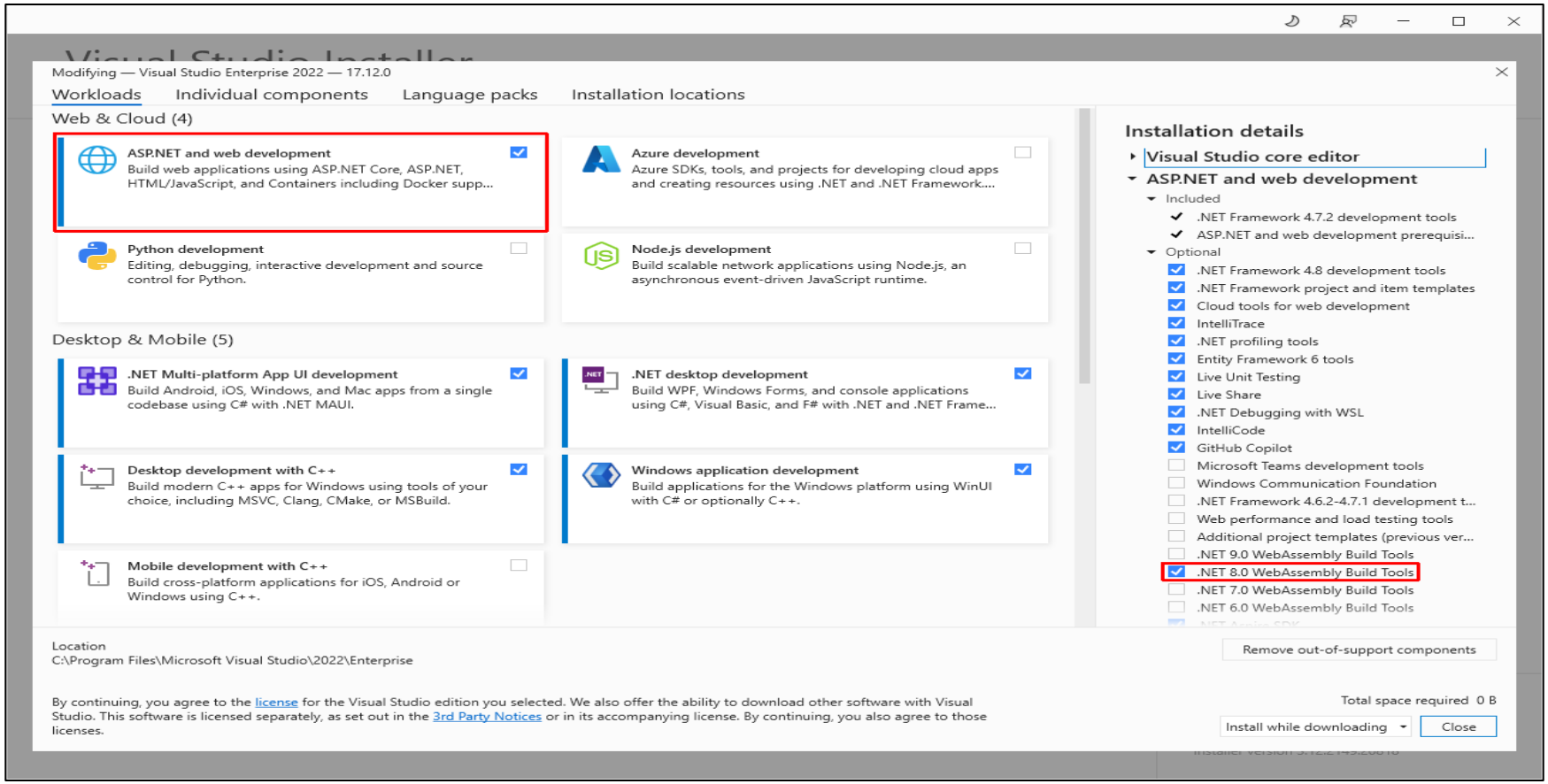
Avalonia UI alapismeretek

Mobil fejlesztés támogatásának telepítése Visual Studioban



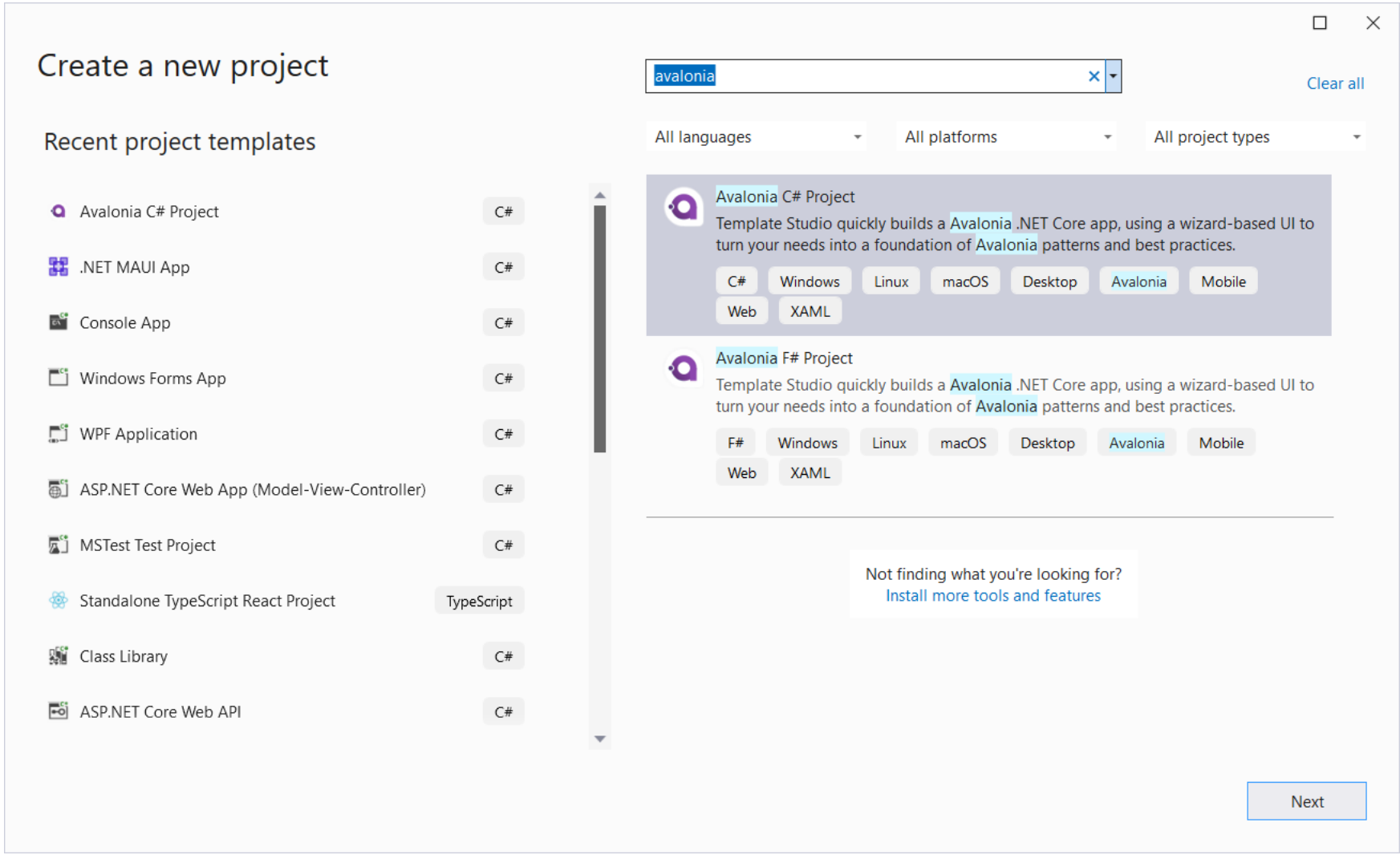
Avalonia UI alapismeretek

WebAssembly fejlesztés támogatásának telepítése Visual Studioban



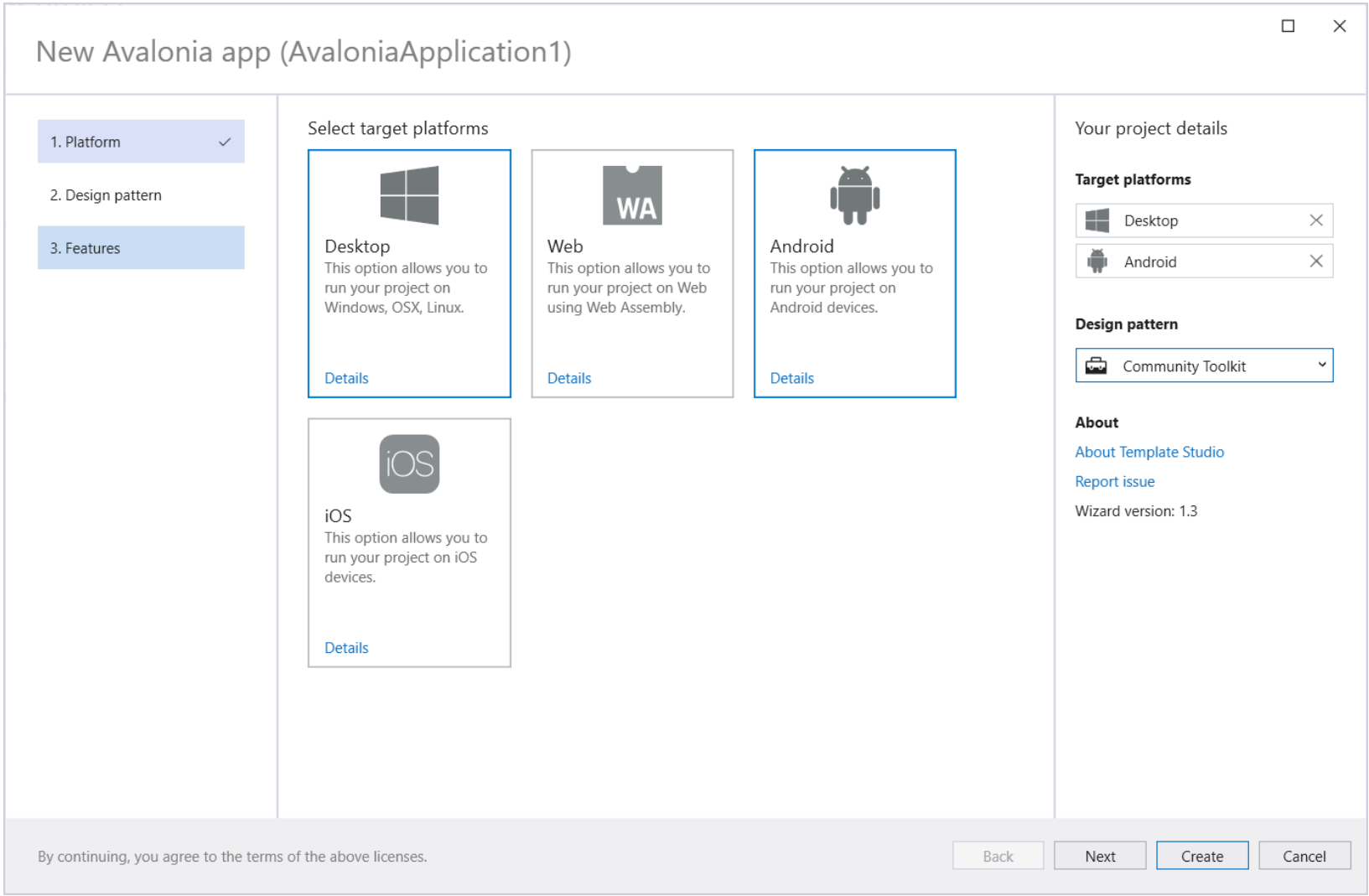
Avalonia UI alapismeretek

Új projekt létrehozása



Avalonia UI alapismeretek

Új projekt létrehozása

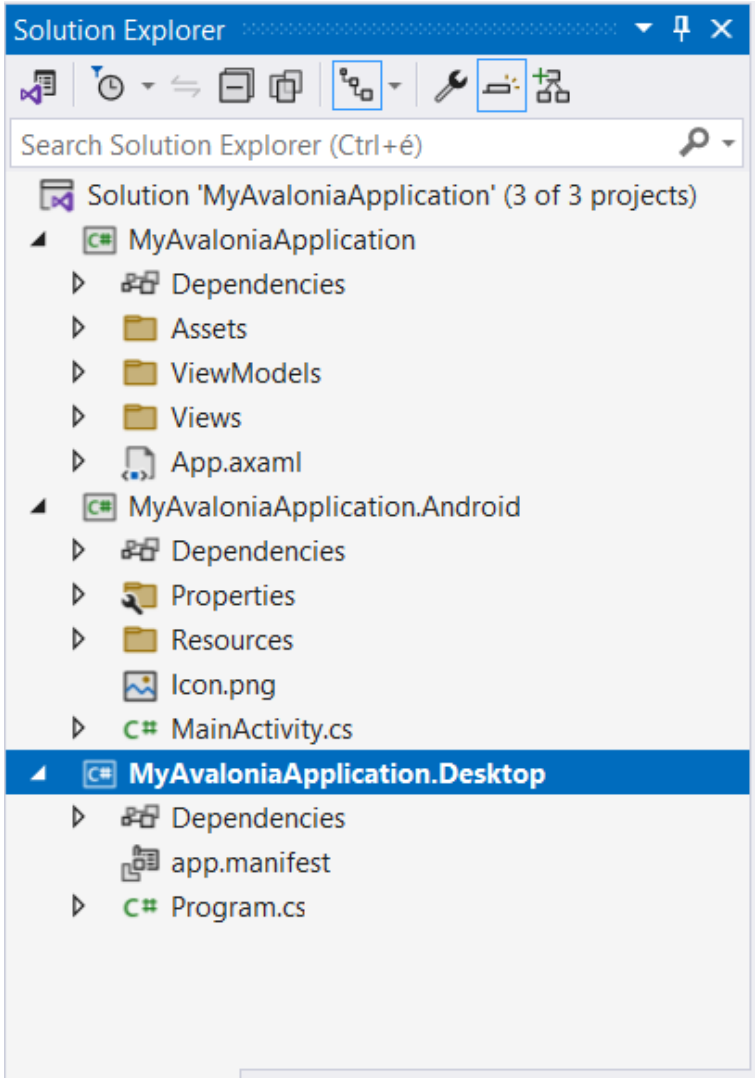


Avalonia UI alapismeretek

Felépítés

Multi-projekt szerkezet

- *.NET Class Library* projekt a cross-platform kódbázisnak
- Platformfüggő projektek (végrehajtható bináris kimenettel) a platformspecifikus kódrészeknek



Avalonia UI alapismeretek

Avalonia alkalmazások felépítése

- Avalonia UI alkalmazások esetén a közös programegységek (osztálykönyvtárak) tartalmazzák
  - a modellt, amely tartalmazza az üzleti logikát, szokványos eszközök segítségével felépítve
  - a nézetmodellt, amelyet az alapvető eszközök segítségével tudunk felépíteni (**`ICommand`**, **`INotifyPropertyChange`**, stb.)
  - a nézetet, amely XAML alapon írunk le, adat- és parancskötés (**`Binding`**) segítségével kapcsoljuk a nézetmodellhez
  - az alkalmazás vezérlését (**`App`**), amely meghatározza a közös viselkedést minden platformon
  - a cross-platform perzisztenciát; vagy egyedi perzisztencia esetén annak interfészét, amely megvalósítása platformonként eltérhet

Avalonia UI alapismeretek

Avalonia alkalmazások felépítése

- Az alkalmazások közös funkcionalitását *.NET Standard Library* segítségével valósíthatjuk meg
  - a közös programegységek is felruházhatóak platformspecifikus jellemzőkkel (kondícionális fordítással vagy a platform futási idejű vizsgálatával)
- A platformspecifikus programegységek (Desktop, Android, iOS, WebAssembly) tovább bővíthetik a közös funkcionalitást
  - tartalmazhatnak egyedi perzisztencia megvalósítást, mivel az adattárolás módja platformonként eltér
  - tartalmazhatnak speciális nézetbeli elemeket, amelyek adott platformon érhetőek csak el, illetve lehetőséget a nézet adaptálására

Avalonia UI alapismeretek

Avalonia grafikus felület felépítése

- Az Avalonia UI alkalmazások egy egységes, *Google Skia* alapú grafikus felülettel rendelkeznek
  - a felületet deklaratív módon, XAML szintaxissal írhatjuk le
    - a fájlok kiterjesztése **`.axaml`**, de a **`.xaml`**-tól való eltérésnek csak technikai oka van (Visual Studio integráció)
  - a felület platformtól függően ablakokból (asztali alkalmazás) vagy nézetekből (mobil alkalmazás) áll
  - amennyiben asztali- és mobilplatformot is támogatunk, a felületet a nézetekben (**`UserControl`**) valósítjuk meg, ezeket ágyazzuk közvetlenül az ablakokba (**`Window`**)
    - így elkerülhető a kódredundancia a nézet rétegben

Avalonia UI alapismeretek

Avalonia grafikus felület felépítése

- Pl. (**`MainView.axaml`**):

```
<UserControl xmlns=https://github.com/avaloniaui
             ...
             x:Class="MyApp.Views.MainView">

    <TextBlock Text="Hello Avalonia UI!"
               HorizontalAlignment="Center"
               VerticalAlignment="Center" />

</UserControl>
```

Avalonia UI alapismeretek

Avalonia grafikus felület felépítése

- Pl. (`MainWindow.axaml`):

```
<Window xmlns="https://github.com/avaloniaui"
        ...
        xmlns:views="clr-namespace:MyApp.Views"
        x:Class="MyApp.Views.MainWindow">

        <views:MainView />

</Window>
```

Avalonia UI alapismeretek

Avalonia grafikus felület felépítése

- Az Avalonia UI felületi vezérlői és lehetőségeik ismerősek lehetnek a WPF keretrendszerből, de eltérések is adódnak.
  - Széles körű beépített vezérlő áll rendelkezésre: https://docs.avaloniaui.net/docs/basics/user-interface/controls/builtin-controls
  - Megjelenítők (**`TextBlock`**, **`Label`**, **`AutoCompleteBox`**, stb.)
  - Nyomógombok (**`Button`**, **`ToggleButton`**, **`RadioButton`**, stb.)
  - Csoportos megjelenítők (**`ListBox`**, **`ItemsControl`**, stb.)
  - Beviteli vezérlők (**`TextBox`**, **`Slider`**, **`Calendar`**, stb.)
  - Elrendezők (**`Canvas`**, **`Grid`**, **`StackPanel`**, **`WrapPanel`**, stb.)

Avalonia UI alapismeretek

Alkalmazás tulajdonságok és kihelyezés

- Az alkalmazások tulajdonságait, képességeit az *alkalmazás leíró* (*application manifest*) segítségével írhatjuk le
  - tartalmazza az alkalmazás nevét, leírását, verzióját és a fejlesztő adatait
  - megadja az engedélyeket a rendszerhez, és más alkalmazásokhoz, pl. internet, kamera, pozicionálás, telefonkönyv, stb.
  - Android esetén az **`AndroidManifest.xml`** fájl, Windows esetén a **`Package.appxmanifest`** fájl, iOS/Mac esetén az **`Info.plist`** fájl tartalmazza a leírást

- A megfelelően konfigurált alkalmazások kihelyezhetőek fizikai eszközökre, illetve elhelyezhetőek a platform alkalmazásboltjában is, ehhez az alkalmazást megfelelő aláírással kell ellátnunk, amely szintén platformspecifikus

Avalonia UI alapismeretek

Példa

Feladat: Készítsünk egy egyszerű számológépet, amellyel a négy alapműveletet végezhetjük el, illetve láthatjuk korábbi műveleteinket is.

- a modell (**`CalculatorModel`**) biztosítja a számológép funkcionalitást, ezt újrahasznosítjuk
- a nézetben (**`MainView`**) elhelyezünk egy rácsot, benne a beviteli mezőt (**`TextBox`**), a gombokat (**`Button`**), valamint a számítások listáját (**`TextBlock`**)
- a gombokhoz közös eseménykezelőt rendelünk (**`ButtonClicked`**), és a gomb szövege alapján döntünk a műveletről
- az esetleges hibákról figyelmeztető üzenetet küldünk (**`MessageBox.Avalonia`** NuGet csomag használatával)

Avalonia UI alapismeretek

MVVM architektúra

- Az Avalonia UI támogatja az MVVM architektúra alapú fejlesztést, így biztosított
  - az adatkötés (**`Binding`**) a nézet oldalon, amelynek megadhatunk tetszőleges forrást (**`DataContext`**)
    - minden vezérlőnek külön is megadható forrás a **`DataContext`** tulajdonság segítségével
    - az elnevezett elemekre (**`x:Name`**) is hivatkozhatunk a kötésben (**`x:Reference`**), pl.:

```
<TextBox x:Name="MyTextBox" />
<Label Text="{Binding
              #MyTextBox.Text.Length}" />
<!-- a címke a szövegdoboz tartalmának
hosszát jelenítse meg -->
```

Avalonia UI alapismeretek

MVVM architektúra

- a változásjelzés (**`INotifyPropertyChanged`**, **`ObservableCollection`**), valamint a parancsvégrehajtás (**`ICommand`**) nézetmodell oldalon
- az alkalmazás vezérlése az alkalmazás (**`App`** osztály) segítségével
  - a nézet adatforrását a **`DataContext`** tulajdonságán keresztül adhatjuk meg
  - a nézet kiválasztása azonban már függ az alkalmazás életciklusától (**`IClassicDesktopStyleApplicationLifetime`** vagy **`ISingleViewApplicationLifetime`**)
    - asztali alkalmazásban tetszőlegesen sok ablakunk lehet, egy fő ablakot kell megadnunk (**`MainWindow`**)
    - mobil alkalmazásban egy nézet látszódik, ezt kell kicserélni, más felületet megjelenítéséhez (**`MainView`**)

Avalonia UI alapismeretek

MVVM Toolkit

- Az MVVM Toolkit az egyik népszerű, MVVM alapú fejlesztést segítő NuGet csomag (*CommunityToolkit.Mvvm*)
  - Használható Avalonia UI-hoz, de WPF-hez is
  - Avalonia UI esetén alapértelmezetten a projekthez adásra került, ha ezt a tervezési mintát választottuk a létrehozáskor
  - Gyakran használt őstípusokat és az MVVM alapú fejlesztést segítő kódgenerátor attribútumokat tartalmaz
- A korábban megismert **`ViewModelBase`** ebben **`ObservableObject`** néven szerepel
  - Továbbra is lehet egy **`ViewModelBase`** osztályunk ebből származtatva, további közös funkcionalitást biztosítva

Avalonia UI alapismeretek

MVVM Toolkit

- A korábban megismert **`DelegateCommand`** itt **`RelayCommand`**
  - elérhető **`RelayCommand`** és **`RelayCommand<T>`** is, pl.:
    ```
    public RelayCommand<int> MyCommand { get; set; }
    ```
  - metódus attribútumaként is alkalmazható, pl.:
    ```
    [RelayCommand] // SomethingCommand parancs
    private void Something() { … }
    ```
    - generált kód:
      ```
      private RelayCommand? somethingCommand;
      public IRelayCommand SomethingCommand =>
        somethingCommand ??= new
          RelayCommand(Something);
      ```
    - az osztályunk szükséges, hogy parciális (**`partial`**) legyen, hogy a generált kódot mellé tudja illeszteni

Avalonia UI alapismeretek

MVVM Toolkit

- A korábban megismert **`OnPropertyChanged()`** helyett **`SetProperty()`** érhető el
  - a **`SetProperty(ref name, value)`** beállítja a **`name`** adattagot, majd kiváltja a **`PropertyChanged`** eseményt rá
  - az **`ObservableProperty`** annotáció adattag attribútumaként is használható, pl.:

```
[ObservableProperty] // Data property
private string _data { … }
```

    - generált kód:

```
public string? Data {
    get => name;
    set => SetProperty(ref _data, value);
}
```

    - az osztályunk szükséges, hogy parciális (**`partial`**) legyen

Avalonia UI alapismeretek

Példa

Feladat: Készítsünk egy egyszerű számológépet, amellyel a négy alapműveletet végezhetjük el, illetve láthatjuk korábbi műveleteinket is.

- valósítsunk meg MVVM architektúrát a nézetmodell kiemelésével
- a nézetmodell (**`CalculatorViewModel`**) tartalmazza az aktuális értéket (**`NumberFieldValue`**), a számítások listáját (**`Calculations`**) és a számítást parancs formájában (**`CalculateCommand`**)
  - a számítási hibákkal kapcsolatosan eseményt küld (**`ErrorOccured`**)
- az alkalmazás példányosítja és összeállítja az alkalmazás rétegeit, és kezeli a számítási hibák eseményeit