

16. Mobilní aplikace, Asynchronní programování

Xamarin

Soubor nástrojů a knihoven pro vývoj aplikací na různé platformy (Android, IOS, Windows).

Vytvoření nativních aplikací pro každou platformu se sdílenou vrstvou s základní logikou (databáze, základní funkcionality, servisní přístup). Nad sdílenou vrstvou je vytvořena nativní aplikace pro každou potřebnou platformu, která používá právě jednu sdílenou logiku.

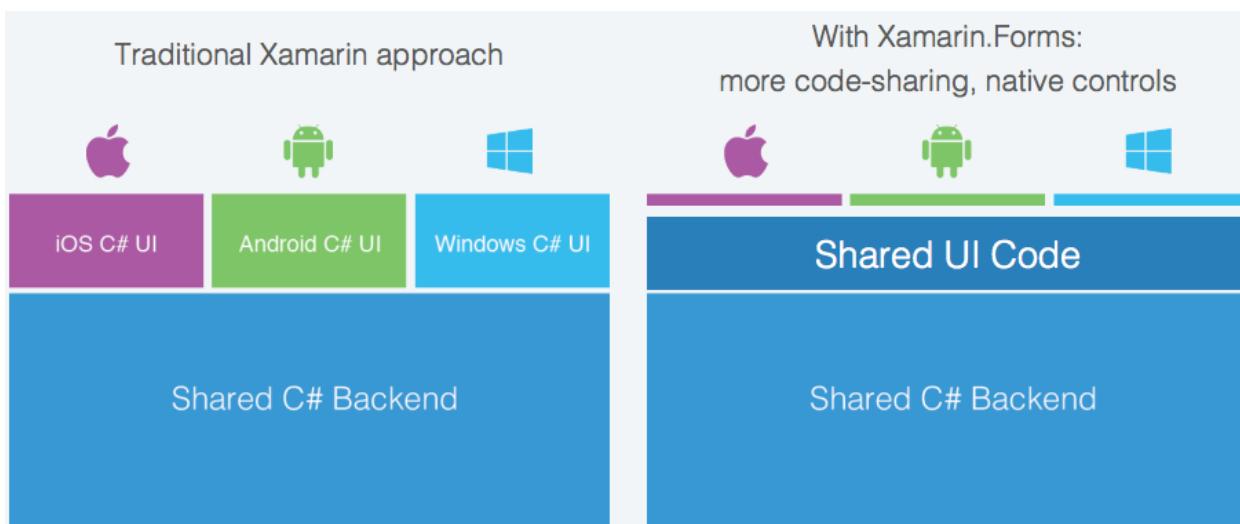
Xamarin.Forms

Xamarin.Forms je UI framework, který umožňuje vytvoření jednotného kódu pro všechny platformy. Využívá k tomu sdílený kód (v C#, XAML), který je později přeložen pro jednotlivé potřeby platforem. Proto můžeme napsat jednu aplikaci, která poběží na jakékoli podporované platformě.

Aplikace má pak jednoduché komponenty, které jsou propojeny s nativními prvky platforem. (NavigationPage bude vypadat jinak na IOS a Android i když bude pro ně jeden kód). Stejně tak i kontrolní prvky (Entry, ImageCell, TextCell, ...)

Základní komponenty:

1. Stránka – ContentPage, NavigationPage, MasterDetailPage, TabbedPage, Templated, Carousel
2. Rozvržení – StackLayout, RelativeLayout, Grid, Flex, AbsoluteLayout, ScrollView, Frame



Proces vývoje mobilní aplikace

Mění se priority v programování a klade se větší důraz na jednoduchost a mobilní platformy (proto Mobile First).

Základní kroky:

1. Koncepce – stanovení strategických cílů k úspěšnému naplnění vašich představ (také je nutný nápad a poptávka)
2. Plán a tým – základní rozpis požadavků, vytvoření týmu a příprava na tvorbu produktu
3. Design – první návrh aplikace se základními funkcemi a stanovení důležitých součástí appky
4. Vývoj – vytvoření první reálné verze aplikace a silného backendu
5. Testy – zjištění chyb a problémů, zajištění pohodlného užívání
6. Zahájení prodeje aplikace + podpora

OUR MOBILE APP DEVELOPMENT PROCESS



Asynchronní programování

Kód píšeme pořád ve stejném stylu jako posloupnost úkolů, co má program vykonat (snadné čtení i při vykonávání některých prací v jiném pořadí). Pokud program narazí na asynchronní metodu, tak vrátí *Task* (reprezentace práce, která ještě potrvá). Toto využijeme v momentě, kdy potřebujeme počkat na nějaký výsledek, ale chceme, aby aplikace stále fungovala jako normálně (např. když čekáme na internetové spojení pro načtení nových dat, tak abychom stále mohli procházet staré informace).

Asynchronní metodu popíšeme slovem *async*. Ta nám umožní použití volání *await*. Tento keyword změní fungování návratové hodnoty. *Task* funguje u asynchronní metody jako *void* u synchronní (může se použít i *void*, vyjímečně).

```
public async Task DoSomethingAsync () {  
    await Task.Delay(1000);  
}
```

Pokud je potřeba návratová hodnota, použije se typ *Task<T>* *T* je návratová hodnota. Asynchronní volání začíná až při zavolení operátoru *await*.

Pro snazší zápis a čtení využíváme Lambda zápis:

```
private async void MetAsync(object s, RoutedEventArgs e) {  
    var Out = await Task.Run(() => { DoSomething();  
})
```

Asynchronní metody používáme na rozdělení dvou hlavních kontextů (UI a Thread pool). Na UI pracujeme převážně s grafickým prostředí a chceme zaručit nepřetržité používání (bez záseků a čekání na výsledky hledání, ...). Naopak na Thread pool chceme naházet všechny logické výpočty a různá načítání dat, aby pracovali zároveň s UI kontextem.

