Aplikace demonstrující nové vlastnosti knihovny XamarinForms

Pavel Šimek

Bakalářská práce
2020

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

Ústav informatiky a umělé inteligence

Akademický rok: 2019/2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Pavel Šimek**
Osobní číslo:	**A17155**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Aplikace demonstrující nové vlastnosti knihovny Xamarin Forms**
Téma práce anglicky:	**An Application for Illustrating New Features of the Xamarin Forms Library**

Zásady pro vypracování

1. Popište současný stav technologií mobilního vývoje.
2. Zaměřte se na framework Xamarin Forms a jeho možnosti.
3. Zpracujte přehled nových vlastnost frameworku jako je Shell, CollectionView a CarouselView a srovnejte jej se staršími postupy.
4. Navrhněte ukázkovou aplikaci s využitím knihovny Xamarin Frorms demonstrující jeho nové vlastnosti.
5. Vytvořte vzorová řešení demonstrující klíčové prvky navržené aplikace.
6. Demonstrujte výsledky.

Rozsah bakalářské práce:
Rozsah příloh:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. HERMES, Dan. Building xamarin.forms mobile apps using xaml: mobile cross-platform xaml and xamarin.forms fundamentals. New York, NY: Springer Science Business Media, 2019. ISBN 978-148-4240-298.
2. BENNETT, Jim. Xamarin in action: creating native cross-platform mobile apps. Shelter Island: Manning, 2018. ISBN 978-1617294389.
3. DANIEL, Steven F. Mastering Xamarin UI Development- Second Edition: Build robust and a maintainable cross-platform mobile UI with Xamarin and C# 7, 2nd Edition. 2nd ed. Packt Publishing, 2018. ISBN 9781788995511.
4. HERMES, Dan. Xamarin Mobile Application Development: Cross-Platform C# and Xamarin.Forms Fundamentals. Apress, 2015. ISBN 978-1484202159.
5. VERSLUIS, Gerald. Xamarin.forms essentials: First Steps Toward Cross-Platform Mobile Apps. New York, NY: Springer Science Business Media, 2017. ISBN 978-148-4232-392.
6. JOHNSON, Paul F. Cross-platform UI Development with Xamarin.Forms. Packt Publishing, 2015. ISBN 978-1784391195.
7. SNIDER, Ed. Mastering Xamarin.Forms – Second Edition: Build rich, maintainable, multi-platform, native mobile apps with Xamarin.Forms, 2nd Edition. 2nd ed. Birmingham: Packt Publishing, 2018. ISBN 978-1-78829-026-5.
8. Xamarin.Forms – Xamarin | Microsoft Docs. [online]. Dostupné z: https://docs.microsoft.com/cs-cz/xamarin/xamarin-forms/

Vedoucí bakalářské práce: **Ing. Erik Král, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání bakalářské práce: **28. listopadu 2019**
Termín odevzdání bakalářské práce: **15. května 2020**

L.S.

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Mgr. Roman Jašek, Ph.D.
ředitel ústavu

Ve Zlíně dne 9. prosince 2019

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 07.08.2020

Pavel Šimek, v.r.
podpis diplomanta

ABSTRAKT

Práce seznamuje s technologiemi mobilního vývoje se zaměřením na multiplatformní vývoj. Dále pojednává o frameworku Xamarin.Forms sloužící pro multiplatformní vývoj a jeho nových funkcích, jako je Shell, CollectionView, CarouselView, jejich použití a rozdílu mezi dříve používanými funkcemi. Poznatky nabyté v části teoretické jsou následně aplikované při tvorbě praktické části, kterou je mobilní, multiplatfromně vyvíjená aplikace s příklady využití zmíněných nových funkcí frameworku Xamarin.Forms.

Klíčová slova: Xamarin, iOS, Android, Xamarin.Forms, CarouselView, CollectionView, Shell, C#, XAML, multiplatformní vývoj

ABSTRACT

This work introduces the technologies of mobile development with focus on cross-platform development. It also deals with Xamarin.Forms framework used for cross-platform developing and its new features such as Shell, CollectionView and CarouselView, their usage and about the difference between those new features and old ones. The knowledge gained in the theoretical part of this thesis are applied in creating of a practical part. Practical part is mobile, cross-platform developed application with examples of the use of Xamarin.Forms framework's new features.

Keywords: Xamarin, iOS, Android, Xamarin.Forms, CarouselView, CollectionView, Shell, C#, XAML, multiplatform development

Rád bych poděkoval **Ing. et Ing. Eriku Králi, Ph.D.** za cenné rady, věcné připomínky, vstřícnost a trpělivost při konzultacích při tvorbě této bakalářské práce. Dále bych chtěl poděkovat rodině a přítelkyni za podporu, sílu a motivaci ke zpracování této práce.

Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Mobilní platformy v posledních letech zažívají velký nárůst uživatelů, a proto se vyvíjí i nové technologie, funkce a postupy, které zvyšují produktivnost práce. Multiplatformní vývoj usnadňuje práci, pokud je potřeba vytvořit totožnou aplikaci pro různé mobilní platformy. Xamarin.Forms je framework, který se věnuje multiplatformnímu vývoji a postupně s dobou se vyvíjí, vylepšuje a zároveň jsou přidávány nové funkce, které práci s ním usnadňují.

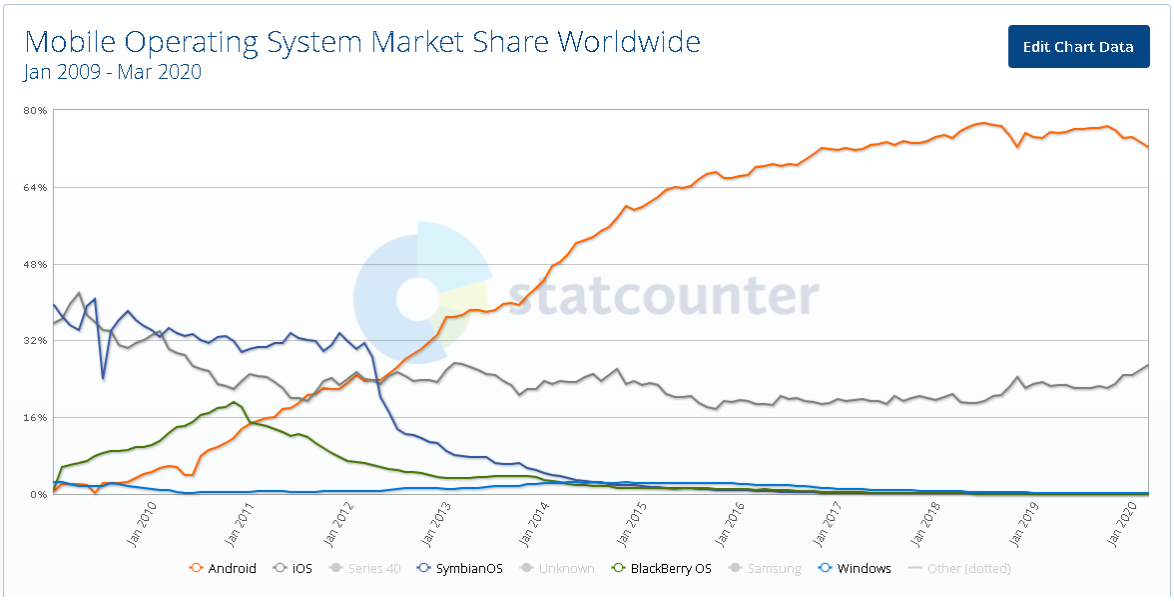
Cílem této práce je seznámit se základními způsoby vývoje software pro mobilní platformy a s novými funkcemi frameworku Xamarin.Forms. Práce se hlavně věnuje funkcím frameworku Xamarin.Forms, které se jmenují Shell, CollectionView, CarouselView, kde se tyto funkce porovnávají se staršími postupy. Poznatky z teoretické části jsou využity při tvorbě multiplatformně vyvíjené mobilní aplikace, která demonstruje využití těchto nových funkcí.

I. TEORETICKÁ ČÁST

1 MOBILNÍ PLATFORMY

Mobilní operační systémy jsou v základě podobné desktopovým operačním systémům. Hlavní rozdíl je, že mobilní operační systémy jsou upravovány pro hardware mobilního zařízení. Dalším rozdílem je ovládání systému. U desktopových operačních systému používáme vstupní zařízení zpravidla myš a klávesnici. Mobilní zařízení jsou uzpůsobena pro ovládání dotykem přes dotykový displej. V posledních letech také roste snaha o rozšíření ovládání mobilního zařízení pomocí hlasu. [1]

Z webové stránky statcounter [2] vyplývá, že v současné době mezi nejrozšířenější mobilní operační systémy patří Android (73,3 %) od Google, iOS (25,89 %) od Apple, které zabírají více než 99 % trhu. Podíl používání mobilních operačních systému ve světě se mění viz obrázek 1. [2]



Obrázek 1 Podíl mobilních operačních systémů na celosvětovém trhu [2]

1.1 Android

Jádro systému Android je Linuxové a nativní aplikace pro tento systém jsou psané v jazyce Java. Android je v dnešní době nejrozšířenějším mobilním operačním systémem a je vyvíjen společností Google. [4]

Operační systém Android je open-source software, tudíž je dostupný pro všechny výrobce mobilních zařízení a díky své flexibilitě je velmi často využíván. Většina výrobců si tovární verzi od Google upraví pro svoje zařízení, takže na čistý Android moc často nenarazíme. [3]

Android se postupně vyvíjí a průběžně přichází aktualizace, které s sebou přinesou i kompletně novou verzi tohoto operačního systému viz obrázek 2.

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.3%
4.1.x	Jelly Bean	16	1.2%
4.2.x		17	1.5%
4.3		18	0.5%
4.4	KitKat	19	6.9%
5.0	Lollipop	21	3.0%
5.1		22	11.5%
6.0	Marshmallow	23	16.9%
7.0	Nougat	24	11.4%
7.1		25	7.8%
8.0	Oreo	26	12.9%
8.1		27	15.4%
9	Pie	28	10.4%

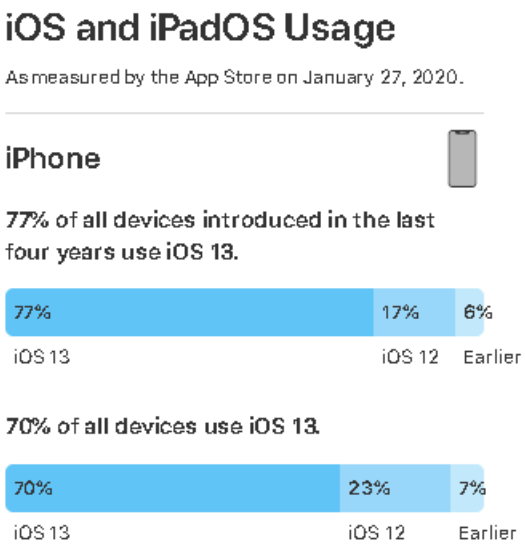
Obrázek 2 Podíl verzí Androidu [6]

1.2 iOS

Mobilní operační systém iOS je vytvořený společností Apple. Tento systém se používá pouze pro zařízení téhle společnosti. Nativní aplikace pro iOS se píší v jazyce Objective-C.

Aplikace pro tento systém se nachází v Apple AppStore a jejich umístění na tuto platformu má přísnější podmínky, než je tomu u operačního systému Android. Pro iOS není možné stahovat aplikace z jiných zdrojů, pokud na zařízení není použita nějaká neoficiální úprava. [4]

iOS je druhým nejrozšířenějším mobilním operačním systémem na trhu a stejně jako v případě systému Android se i operační systém průběžně vyvíjí. Využití verzí systému iOS viz obrázek 3.



Obrázek 3 Podíl verzí iOS [7]

2 TECHNOLOGIE MOBILNÍHO VÝVOJE

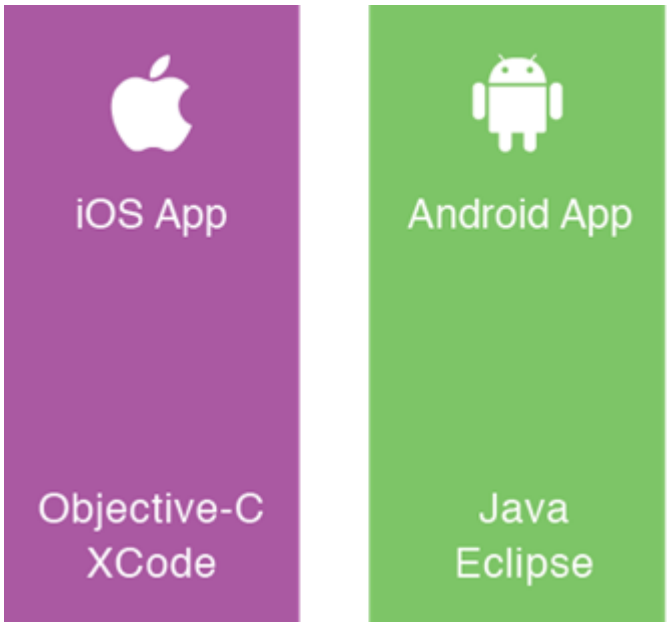
Mobilní aplikace se mohou vyvíjet různými způsoby a každý z těchto způsobů má kladné i záporné stránky. V následující kapitole si blíže popíšeme určité technologie mobilního vývoje a uvedeme si kladné i záporné stránky daného způsobu vývoje.

2.1 Nativní vývoj aplikací

Nativní vývoj mobilních aplikací je způsob vývoje, který se zaměřuje čistě na jednu mobilní platformu.

Tento vývoj pracuje s programovacím jazykem, který je určený pro daný operační systém. V případě Androidu je to Java, nebo Kotlin, pro iOS je tomu Objective-C a programovací jazyk Swift. [9]

Pro nativní vývoj mobilních aplikací jsou přístupné různé sady vývojových nástrojů (SDK) a vývojových prostředí (IDE) viz obrázek 4, které vývojářům usnadňují práci a díky těmto sadám je možné vytvářet složitější aplikace jednodušším způsobem. Tyto sady jsou poskytovány tvůrci daných platforem.



Obrázek 4 Programovací jazyky a IDE pro Android a iOS [11]

2.1.1 Výhody nativního vývoje mobilních aplikací

Hlavní výhodou nativního vývoje mobilních aplikací je využití všech možností hardwarového vybavení zařízení a operačního systému. Lze získat přístup k paměti zařízení, kameře, k poloze zařízení, mikrofonu, polohovým senzorům, čtečce otisku prstů. [9]

Navíc díky nativnímu vývoji je aplikace schopna uživateli zobrazovat různá upozornění, používat vibrace a zvuky. Využitím těchto funkci je aplikace schopna dopřát uživateli daného mobilního zařízení kvalitnější a příjemnější zážitek užíváním dané aplikace.

Další velkou výhodou je to, že pro vyvíjenou aplikaci nejsou limity ve výkonu a rychlosti. To umožňuje vyvíjet složité aplikace, které jsou náročné na grafiku, a rychlost zařízení, například mobilní hry. K funkčnosti aplikace není potřeba internetového připojení. [10]

2.1.2 Nevýhody nativního vývoje mobilních aplikací

Každá mobilní platforma používá jiný jazyk a využitím nativního vývoje bude muset vzniknout více aplikací, což se promítne na ceně dané aplikace. [10]

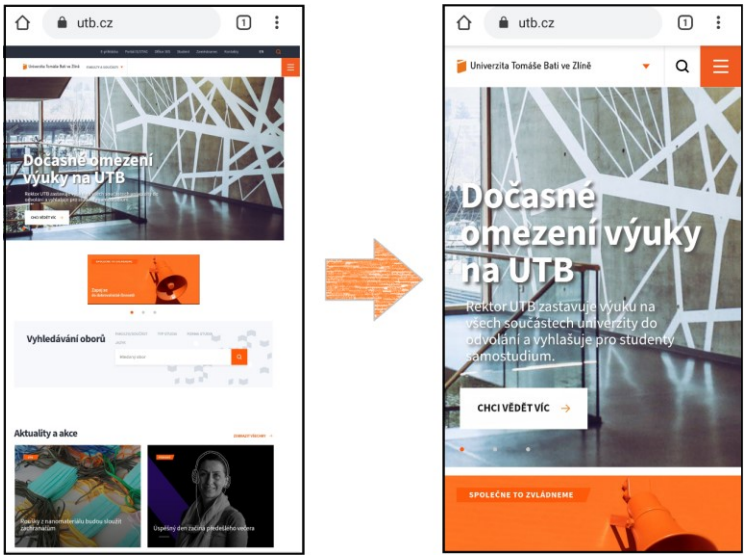
Jak plyne z předchozí věty, tak se nedá sdílet kód aplikace, což výrazně zpomalí celý průběh vývoje. Taktéž se tato skutečnost projeví na testování aplikace, jelikož je potřeba otestovat mnohem více kódu. [9]

2.2 Progresivní webové aplikace

Progresivní webové aplikace (PWA) jsou aplikace připravené pro použití na téměř každém mobilním zařízení s přístupem k internetu. K těmto aplikacím se přistupuje pomocí internetového prohlížeče v mobilním zařízení, jako k normálním webovým stránkám na desktopovém zařízení. Rozdíl je pouze ten, že webová aplikace je přizpůsobena menšímu displeji a zobrazuje obsah webu jiným způsobem viz obrázek 5. [8]

Dokonce se může stát, že některé prvky ze stránky zmizí pro lepší uživatelský zážitek. Webové aplikace se tvoří převážně pomocí HTML5, CSS a JavaScriptu.

Tyto aplikace se nenachází v oficiálním obchodu s aplikacemi (AppStore, nebo Google Play). Během používání této mobilní aplikace se může zobrazit tlačítko stáhnout. To ve skutečnosti jen vytvoří záložku webové stránky a zobrazí ji v menu zařízení, kde pro přístup k aplikaci stačí kliknout na ikonku dané záložky. [8]



Obrázek 5 Rozdíl mezi standardním a přizpůsobeným zobrazením webové stránky [12]

2.2.1 Výhody webové aplikace

Webové aplikace nevyžadují tolik plánování, vývoje a testování, jako je tomu u nativních aplikací. Vývoj je mnohem rychlejší a díky tomu stojí méně peněz. Spuštění těchto aplikací je možné z téměř každého zařízení, takže není potřeba aplikaci vyvíjet pro každou platformu zvlášť. [10]

2.2.2 Nevýhody webové aplikace

Mezi nevýhody jednoznačně patří nemožnost využití většiny hardwarových funkcí zařízení. Na rozdíl od nativního vývoje aplikace nemá přístup k těmto funkcím a díky tomu není schopna uživateli poskytnout takový komfort. Problém nastává i v náročnosti aplikace, kde je potřeba, aby aplikace nebyla tak náročná, jelikož se jí dostává menšího výkonu než u aplikací psaných nativně. [10]

2.3 Hybridní způsob vývoje

Hybridní vývoj je něco mezi Webovými aplikacemi a nativním způsobem vývoje. Ve výsledku je to webový obsah zabalen v nativním prostředí. Hlavní část aplikace je psána např. pomocí HTML5, CSS a JavaScriptu. Díky nativně psané části aplikace, která je specifická pro dané platformy, může mít aplikace přístup k některým funkcím zařízení (kamera, GPS, …), jako u nativního způsobu vývoje. Aplikace vytvořené tímto způsobem můžeme naleznout v oficiálních obchodech daných platforem (AppStore, Google Play), tudíž se po instalaci tváří jako nativně vytvořená aplikace. [16]

2.3.1 Výhody hybridního způsobu vývoje

Sdílení části kódu urychlí vývoj, testování a se rovná nižší cenně finální aplikace. Vývoj je celkově jednodušší a rychlejší právě díky využití webových prvků.

2.3.2 Nevýhody hybridního způsobu vývoje

Nedosahují stejného výkonu, jako je tomu u aplikací nativních, nebo aplikací vytvořených multiplatformním způsobem (multiplatformnímu způsobu vývoje se věnuje kapitola 3) a to může některé uživatele od používání takovéto aplikace odradit.

2.4 Shrnutí

Každý ze zmíněných způsobu vývoje má klady a zápory. Pro výběr nejvhodnějšího způsobu vývoje je důležité vzít v potaz pro jakou skupinu uživatelů je aplikace zaměřena, potřebný výkon aplikace, cena, doba vývoje a nutnost využití hardwarové příslušenství v telefonu. Tabulka 1 shrnuje základní informace o způsobech mobilního vývoje.

Tabulka 1 Srovnání nativního, hybridního a webového vývoje aplikací [8]

	Nativní	**Hybridní**	**PWA**
Cena	Nejdražší	Střední	Nejlevnější
Uživatelé	Pouze platforma, pro kterou byla daná aplikace vytvořena	Může být jednoduše přenesena na více operačních systémů	Může být spuštěna v každém zařízení, které má internetový prohlížeč a přístup k internetu
Vhodné pro	Náročné aplikace, například hry	Méně náročné aplikace	Responzivní weby, které jsou pro zařízení velmi málo náročné
UX	Skvělé	Dobré	Velmi omezené
Funkčnost	Aplikace je schopna využít všechny prvky, které zařízení obsahuje	Aplikace má přístup k některým prvkům zařízení (kamera, GPS)	Aplikace nemá přístup k žádným prvkům zařízení

3 MULTIPLATFORMNÍ VÝVOJ

Multiplatformní vývoj mobilních aplikací (cross-platform mobile development) je způsob vývoje, kdy jsou aplikace pro různé platformy psané v jednom jazyce, pomocí sdíleného kódu. Vývoj tímto způsobem je efektivní, protože není potřeba psát pro každou platformu vlastní kód. Při vytváření aplikace se zdrojový kód překládá do kódu nativního jazyka pro danou platformu.

Kód, který není možné sdílet tímto způsobem mezi všechny platformy musí být napsán v nativním jazyce pro danou platformu. Je možné sdílet přibližně 80 % kódu mezi různé platformy. Po překladu aplikace pro danou platformu vznikají spustitelné soubory, které mohou být nahrány do oficiálních obchodů s aplikacemi. [10]

Mezi jedny z nejčastěji používaných nástrojů pro multiplatformní vývoj jsou Xamarin od Microsoftu, React Native z dílny společnosti Facebook a Flutter vyvíjený firmou Google. [14]

Každý z těchto nástrojů používá jiný programovací jazyk. Pomocí React Native se kód píše v jazyce JavaScript, Xamarin využívá programovací jazyk C# a pro Flutter je tomu Dart. [17]

Srovnání zmíněných nástrojů viz tabulka 2.

3.1 Srovnání nástrojů

Tahle práce se dále věnuje pouze Xamarinu a jeho části Xamarin.Forms.

Xamarin má ze všech zmíněných nástrojů nejvyšší znovupoužitelnost kódu až 96 %. Což výrazným způsobem ovlivní rychlost vývoje aplikace pro všechny platformy. [13]

Tabulka 2 Porovnání nástrojů pro multiplatformní vývoj [15]

	Xamarin.Forms	**React Native**	**Flutter**
Znovupoužitelnost kódu	Až 96 %	Až 90 %	Až 90 %
Výkon aplikace	Střední, téměř stejný jako nativní aplikace	Téměř stejný jako nativní aplikace	Téměř stejný jako nativní aplikace
Programovací jazyk	C#	JavaScript	Dart

3.2 Výhody multiplatformního mobilního vývoje

Aplikace je schopna oslovit velké množství uživatelů, díky tomu, že je přístupná na více platformách. Jednotný kód aplikace zjednoduší vývoj a zároveň sníží cenu a celkový čas potřebný pro vývoj aplikace na požadované platformy.

Vysoká část sdíleného kódu, až 96 % při použití nástroje Xamarin.Forms. [13]

Méně kódu znamená méně testování a jednodušší udržování a úpravy aplikace. Aplikace je schopna dosahovat velmi vysokého výkonu, téměř jako nativně psané aplikace. [10]

3.3 Nevýhody multiplatformního mobilního vývoje

Sdílení kódu není 100 %, tudíž je potřeba určitou část napsat v nativním jazyce pro každou platformu zvlášť. Uživatel nezíská veškerý komfort, jako při používání nativně psaných aplikací.

Nejsou přístupny všechny funkce mobilního operačního systému a hardwarového vybavení zařízení. Dostupné funkce závisí na možnostech použitého nástroje. Výkon, i přes to, že je uvedený jako výhoda, tak se dá taky počítat jako nevýhoda, protože není stejný, jako je tomu u nativních aplikací. [10]

4 XAMARIN

Obrázek 6 Logo společnosti Xamarin [17]

Společnost Xamarin, logo společnosti viz obrázek 6, byla založena v roce 2011 Miguelem de Icaza, nyní však spadá pod společnost Microsoft. Společnost nabízí stejnojmenný produkt Xamarin, který je open source platformou a slouží k tvorbě multiplatformních aplikací. [19]

Tvoří abstraktní vrstvu, která řídí komunikaci sdíleného kódu s různými mobilními platformami. Xamarin umožňuje sdílet průměrně 80% kódu mezi platformy, což umožňuje vývojářům psát aplikaci v jednom jazyce, nebo používat již napsaný kód. Zároveň aplikace používající Xamarin dosahují vysokého, až nativního výkonu. Díky své výkonnosti je platforma Xamarin cílena na náročnější, nebo business aplikace. [18]

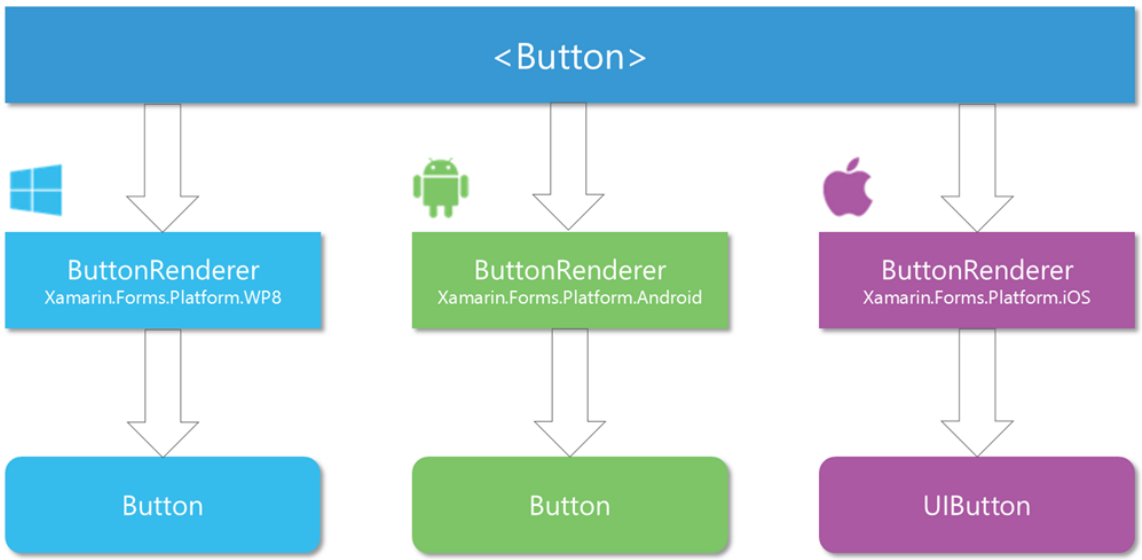
Xamarin aplikace je možno kompilovat do nativních balíčků aplikací:

Soubory .apk pro platformu Android a .ipa pro iOS.

4.1 Xamarin.Forms

Je open-source framework určený k tvorbě UI. Umožňuje nativní vývoj aplikací pro různé platformy najednou a to s minimem dalších úprav pro daný operační systém. Tento Framework využívá programovací jazyky XAML a C# . NET. XAML slouží k definování uživatelského rozhraní, C# .NET je použit na logickou část aplikace. [21]

S použitím frameworku Xamarin.Forms lze sdílet až 96% kódu, který je společný pro všechny platformy. [13]



Obrázek 7 Ukázka renderování tlačítka [20]

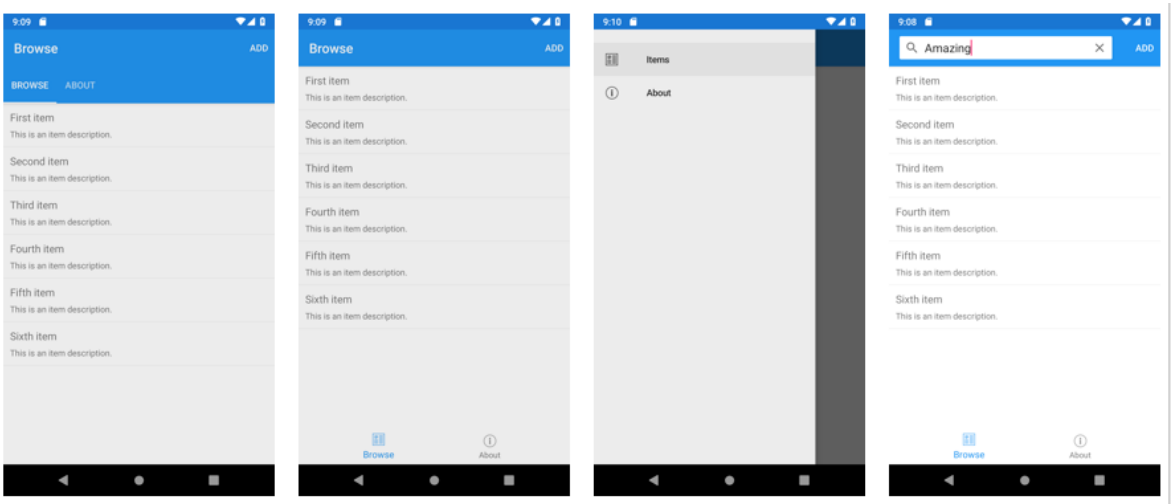
V obrázku 7 je uvedeno, jakým způsobem pracuje mapování komponent aplikace. Každá komponenta je renderována pomocí třídy Renderer do nativní verze pro danou platformu. Lze vytvářet komponenty nové vytvořením nových rendererů a nebo do již existujících rendererů přidávat různá rozšíření pro danou, již vytvořenou komponentu. [20]

5 NOVÉ VLASTNOSTI FRAMEWORKU XAMARIN.FORMS

V průběhu vývoje Xamarin.Forms jsou přidávány nové funkce a vlastnosti za účelem zrychlení práce a zjednodušení psaní kódu pro vývojáře. Mezi tyto funkce patří například Shell, CollectionView a CarouselView. Zmíněné funkce budou blíže popsány a zároveň proběhne srovnání s postupy, které byly používány v minulosti.

5.1 Shell

Funkce Xamarin.Forms Shell byla představena ve verzi Xamarin.Forms 4.0. Znamená značné zjednodušení a nahrazení starých funkcí. Tyto staré funkce bylo potřeba používat v kombinacích, aby bylo dosaženo požadovaného výsledku. Shell tohle celé zjednodušilo a celkově zrychlilo práci při vytváření navigace mezi stránkami aplikace. Ukázka možností Shell viz obrázek 8.



Obrázek 8 Možnosti Shell [22]

Shell představuje jednoduchý kontejner, který nahrazuje složitou práci s funkcemi jako jsou TabbedPage, NavigationPage a MasterDetailPage. Jediné, co je potřeba udělat je vytvořit Xaml soubor a začínat aplikaci Shellem – příkaz: <Shell></Shell>. Dále se jen přidávají požadované stránky. Lze nastavit více způsobů, jakými se mezi různými stránkami pohybovat. [22]

Routování v aplikaci přiřazuje sám Shell ke každému použitému elementu. Nicméně tyto Routes lze přejmenovat jednoduchým přidáním vlastnosti Route. [22] Díky deklaraci vlastnosti Route pro daný element, je možné se kdekoliv v aplikaci na dané stránky odkázat. Příklad kódu [22]: *await Shell.Current.GoToAsync("//contact");*

5.1.1 MasterDetailPage

Jedna z možností, jak menu vytvořit je pomocí funkcí <MasterDetailPage>, <MasterDetailPage.Master> a <MasterDetailPage.Detail>, <ContentPage>, <StackLayout>, do kterého se vloží list položek, které chceme v menu mít. Dále je potřeba funkce na sebe navázat. Tento způsob je i s příchodem Shellu podporovaný, nicméně další vývoj bude soustředěn pro Shell. Příklad níže.

```
<MasterDetailPage xmlns="http://xamarin.com/schemas/2014/forms"
            xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
            xmlns:d="http://xamarin.com/schemas/2014/forms/design"
            xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
            mc:Ignorable="d"
            xmlns:views="clr-namespace:App6.Views"
            x:Class="App6.Views.MainPage">

    <MasterDetailPage.Master>
        <views:MenuPage />
    </MasterDetailPage.Master>

    <MasterDetailPage.Detail>
        <NavigationPage>
            <NavigationPage.Icon>
                <OnPlatform x:TypeArguments="FileImageSource">
                    <On Platform="iOS" Value="tab_feed.png"/>
                </OnPlatform>
            </NavigationPage.Icon>
            <x:Arguments>
                <views:ItemsPage />
            </x:Arguments>
        </NavigationPage>
    </MasterDetailPage.Detail>

</MasterDetailPage>
```

Obrázek 9 MasterDetailPage pro vytvoření menu [23]

V obrázku 9 je ukázáno, jakým způsobem je nutné nastavit MasterDetailPage, aby menu bylo funkční. Už v této ukázce je napsáno více kódu, než je tomu při použití Shell a to není vše, co je potřebné. Použití Shell

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:d="http://xamarin.com/schemas/2014/forms/design"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006
             mc:Ignorable="d"
             x:Class="App6.Views.MenuPage"
             Title="Menu">

    <StackLayout VerticalOptions="FillAndExpand">
        <ListView x:Name="ListViewMenu"
                  HasUnevenRows="True">
            <d:ListView.ItemsSource>
                <x:Array Type="{x:Type x:String}">
                    <x:String>Item 1</x:String>
                    <x:String>Item 2</x:String>
                </x:Array>
            </d:ListView.ItemsSource>
            <ListView.ItemTemplate>
                <DataTemplate>
                    <ViewCell>
                        <Grid Padding="10">
                            <Label Text="{Binding Title}" d:Text="{Binding .}"
FontSize="20"/>
                        </Grid>
                    </ViewCell>
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
    </StackLayout>
```

Obrázek 10 ContentPage pro vytvoření menu [23]

Obrázek 10 zobrazuje možnost nastavení zobrazení položek v menu.

```
menuItems = new List<HomeMenuItem>
            {
                new HomeMenuItem {Id = MenuItemType.Browse, Title="Browse" },
                new HomeMenuItem {Id = MenuItemType.About, Title="About" }
            };

            ListViewMenu.ItemsSource = menuItems;

            ListViewMenu.SelectedItem = menuItems[0];
            ListViewMenu.ItemSelected += async (sender, e) =>
            {
                if (e.SelectedItem == null)
                    return;

                var id = (int)((HomeMenuItem)e.SelectedItem).Id;
                await RootPage.NavigateFromMenu(id);
            };
```

Obrázek 11 Vytvoření listu pro položky v menu [23]

V obrázku 11 je uvedena možnost vytvoření listu položek do menu.

```
public partial class MainPage : MasterDetailPage
    {
        Dictionary<int, NavigationPage> MenuPages = new Dictionary<int, NavigationPage>();
        public MainPage()
        {
            InitializeComponent();

            MasterBehavior = MasterBehavior.Popover;

            MenuPages.Add((int)MenuItemType.Browse, (NavigationPage)Detail);
        }

        public async Task NavigateFromMenu(int id)
        {
            if (!MenuPages.ContainsKey(id))
            {
                switch (id)
                {
                    case (int)MenuItemType.Browse:
                        MenuPages.Add(id, new NavigationPage(new ItemsPage()));
                        break;
                    case (int)MenuItemType.About:
                        MenuPages.Add(id, new NavigationPage(new AboutPage()));
                        break;
                }
            }

            var newPage = MenuPages[id];

            if (newPage != null && Detail != newPage)
            {
                Detail = newPage;

                if (Device.RuntimePlatform == Device.Android)
                    await Task.Delay(100);

                IsPresented = false;
            }
        }
    }
```

Obrázek 12 Kód na pozadí pro menu [23]

Obrázek 12 popisuje způsob, kterým je tvořené menu a hlavní stránka aplikace.

Tento složitý a časově náročný postup lze nahradit právě funkcí Shell. Shell vyžaduje, oproti příkladu postupu výše, velmi málo kódu a tím dělá aplikaci jednodušší, přehlednější. Příklady kódů s použitím Shell viz části FlyoutItem a Tabs.

5.1.2 FlyoutItem

Pokud je cílem vytvořit postranní menu, neboli takzvané hamburger menu, přidáme mezi <Shell></Shell> jednoduchý kód, který nám zajistí položku v daném menu. Tento kód je <FlyoutItem></FlyoutItem> s vlastnostmi pro název položky a ikonku. Mezi začátek a konec FlyoutItem se vloží <ShellContent> s vlastností ContentTemplate, jehož práce je nastavení přesměrování na požadované zobrazení v aplikaci. Použitím tohoto jednoduchého způsobu lze vytvořit menu viz příklad [22]:

```
<Shell>
    <FlyoutItem Title="Home" Icon="home.png">
        <ShellContent ContentTemplate="{DataTemplate pages:HomePage}"/>
    </FlyoutItem>
    <FlyoutItem Title="Library" Icon="library.png">
        <ShellContent ContentTemplate="{DataTemplate pages:LibraryPage}"/>
    </FlyoutItem>
    <FlyoutItem Title="Contact" Icon="contact.png">
        <ShellContent ContentTemplate="{DataTemplate pages:ContactPage}"/>
    </FlyoutItem>
</Shell>
```

V kódu je uveden jednoduchý postup pro vytvoření postranního menu se třemi položkami. Tyto položky jsou Home, Library a Contact. Obrázek13 zobrazuje výsledek kódu uvedeného v příkladu

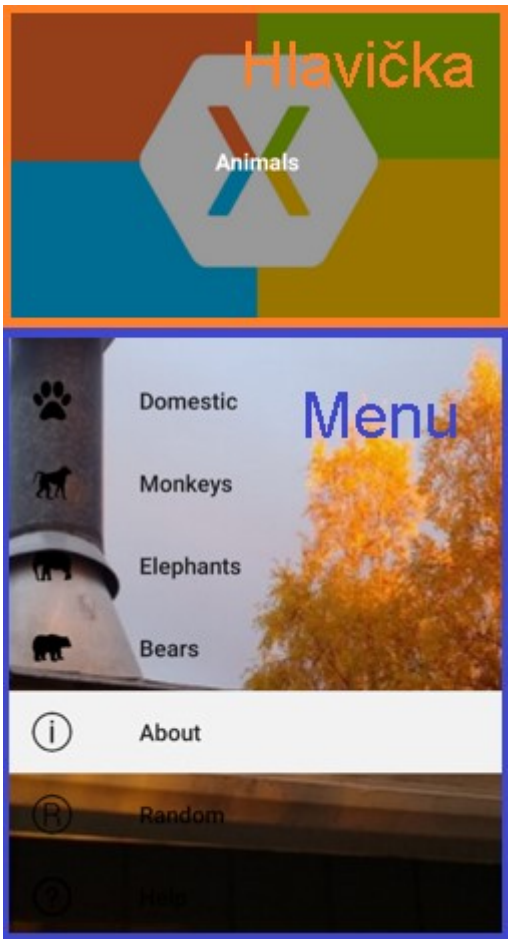
Obrázek 13 Postranní menu vytvořené pomocí Shell [22]

S použitím flyout lze definovat i hlavičku, či obrázek do pozadí celého menu. Zobrazeno viz obrázek 14. Pro přidání libovolného obrázku do pozadí menu je za potřebí přidat mezi vlastnosti Shell následující kód [24]:

FlyoutBackgroundImage="photo.jpg"

FlyoutBackgroundImageAspect="AspectFill"

FlyoutBackgroundImage nastaví požadovaný obrázek jako pozadí menu. FlyoutBackgroundImageAspect nastaví, jakým způsobem se má obrázek v menu vyplnit.



Obrázek 14 Popis menu použitím Flyout [24]

5.1.2.1 Některé z vlastností tříd FlyoutItem a Tabs [24]:

FlyoutDisplayOptions: typu FlyoutDisplayOptions definuje způsob zobrazení položky a jejích podřízených v postranním menu. Výchozí hodnota je AsSingleItem.

CurrentItem: typu Tab, vybraná položka.

Items: typ IList <Tab>, definuje všechny karty v rámci FlyoutItem.

FlyoutIcon: typu ImageSource, ikona použitá pro položku. Pokud tato vlastnost není nastavena, použije se hodnota vlastnosti **Icon**.

Icon: typu ImageSource, definuje ikonu, která se má zobrazit v částech, které nejsou Flyout.

Title: typ string, je název, který se má zobrazit v uživatelském rozhraní.

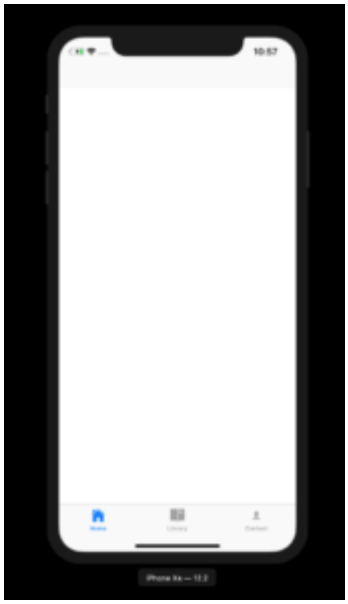
Route: typ string, řetězec použitý k adresování položky.

5.1.3 Tabs

Elementem <TabBar> lze vytvořit menu na spodní části obrazovky viz obrázek 15. Použitím <TabBar> elementu mezi <Shell> se vytvoří prostor pro definování položek v menu. Položky lze definovat pomocí stejného kódu, jako tomu bylo u FlyoutItem, tedy elementem <ShellContent> s jediným rozdílem a to použití elementu <Tab> namísto <FlyoutItem>.

Implenetace pro jednu z možností využití <TabBar> [22]:

```
<Shell>
    <TabBar>
        <Tab Title="Home" Icon="home.png">
            <ShellContent ContentTemplate="{DataTemplate pages:HomePage}"/>
        </Tab>
        <Tab Title="Library" Icon="library.png">
            <ShellContent ContentTemplate="{DataTemplate pages:LibraryPage}"/>
        </Tab>
        <Tab Title="Contact" Icon="contact.png">
            <ShellContent ContentTemplate="{DataTemplate pages:ContactPage}"/>
        </Tab>
    </TabBar>
</Shell>
```

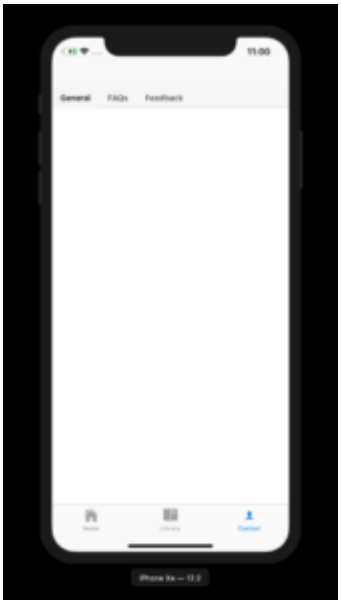


Obrázek 15 Zobrazení využití TabBar s jedním ShellContent [22]

Pokud je v jednom elementu <Tab> přidáno více elementů <ShellContent> vytvoří se v dané záložce menu i ve vrchní části obrazovky viz obrázek 16.

Další možnost implementace [22] s využitím <TabBar>. Rozdíl mezi těmito možnostmi je takový, že kód následující příklad zobrazí navíc tři tlačítka v horní části obrazovky pro položku Contact. Tlačítka v horní části obrazovky se jmenují General, FAQs a Feedback, viz. obrázek 16:

```
<Shell>
    <TabBar>
        <Tab Title="Home" Icon="home.png">
            <ShellContent ContentTemplate="{DataTemplate pages:HomePage}"/>
        </Tab>
        <Tab Title="Library" Icon="library.png">
            <ShellContent ContentTemplate="{DataTemplate pages:LibraryPage}"/>
        </Tab>
        <Tab Title="Contact" Icon="contact.png">
            <ShellContent Title="General" ContentTemplate="{DataTemplate
pages:ContactPage}"/>
            <ShellContent Title="FAQs" ContentTemplate="{DataTemplate
pages:FAQsPage}"/>
            <ShellContent Title="Feedback" ContentTemplate="{DataTemplate
pages:FeedbackPage}"/>
        </Tab>
    </TabBar>
</Shell>
```



Obrázek 16 Zobrazení využití TabBar s více ShellContent v jednom Tab [22]

5.1.4 ShellContent

Slouží k určení, co bude daná položka v menu dále zobrazovat. Této třídě je možné nastavit některé z vlastností popsané níže.

5.1.4.1 Některé z vlastností třídy ShellContent [22]:

Content: typ objektu, obsah ShellContentu.

ContentTemplate: typ DataTemplate, šablony používané k dynamickému nafouknutí obsahu ShellContent.

FlyoutIcon: typu ImageSource, definuje ikonu, která se zobrazí v plovoucí nabídce.

Icon: typu ImageSource definuje ikonu, která se má zobrazit v částech chromu, které nejsou plovoucí nabídkou.

MenuItems: typu MenuItemCollection, položky nabídky, které se mají zobrazit v plovoucí nabídce, když je tato ShellContent prezentovanou stránkou.

5.2 CollectionView

S 4.3 verzí Xamarin.Forms přichází stabilní verze CollectionView. CollectionView je výkonné a flexibilní zobrazení pro prezentování seznamů dat pomocí různých specifikací rozložení. Objekt se naplní daty nastavením jeho vlastnosti „ItemsSource“ na libovolnou kolekci, která implementuje IEnumerable. [25]

V následujícím kódu je zobrazeno, jakým způsobem lze přidávat položky v jazyce XAML do CollectionView.ItemsSource pomocí pole stringů [26]:

```
<CollectionView>
  <CollectionView.ItemsSource>
    <x:Array Type="{x:Type x:String}">
      <x:String>Baboon</x:String>
      <x:String>Capuchin Monkey</x:String>
      <x:String>Blue Monkey</x:String>
      <x:String>Squirrel Monkey</x:String>
      <x:String>Golden Lion Tamarin</x:String>
      <x:String>Howler Monkey</x:String>
      <x:String>Japanese Macaque</x:String>
    </x:Array>
  </CollectionView.ItemsSource>
</CollectionView>
```

Stejná data lze implementovat i pomocí kódu v jazyce C# [26]:

```
CollectionView collectionView = new CollectionView();
collectionView.ItemsSource = new string[]
{
    "Baboon",
    "Capuchin Monkey",
    "Blue Monkey",
    "Squirrel Monkey",
    "Golden Lion Tamarin",
    "Howler Monkey",
    "Japanese Macaque"
};
```

Pokud není změněno původní nastavení CollectionView, zobrazí se položky pouze ve svislém seznamu.

Příklad vázání dat v CollectionView je provedeno v XAML [26]:

```
<CollectionView ItemsSource="{Binding Monkeys}" />
```

Ekvivalentní kód v C# [26]:

```
CollectionView collectionView = new CollectionView();

collectionView.SetBinding(ItemsView.ItemsSourceProperty, "Monkeys");
```

Nastavit vzhled každé položky v CollectionView lze nastavením CollectionView.ItemTemplate na DataTemplate. Elementy specifikované v DataTemplate

definují vzhled každé položky v seznamu. Viz příklad psaný v programovacím jazyce XAML [26]:

```
<CollectionView ItemsSource="{Binding Monkeys}">
    <CollectionView.ItemTemplate>
        <DataTemplate>
            <Grid Padding="10">
                <Grid.RowDefinitions>
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                </Grid.RowDefinitions>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="Auto" />
                    <ColumnDefinition Width="Auto" />
                </Grid.ColumnDefinitions>
                <Image Grid.RowSpan="2"
                       Source="{Binding ImageUrl}"
                       Aspect="AspectFill"
                       HeightRequest="60"
                       WidthRequest="60" />
                <Label Grid.Column="1"
                       Text="{Binding Name}"
                       FontAttributes="Bold" />
                <Label Grid.Row="1"
                       Grid.Column="1"
                       Text="{Binding Location}"
                       FontAttributes="Italic"
                       VerticalOptions="End" />
            </Grid>
        </DataTemplate>
    </CollectionView.ItemTemplate>
    ...
</CollectionView>
```

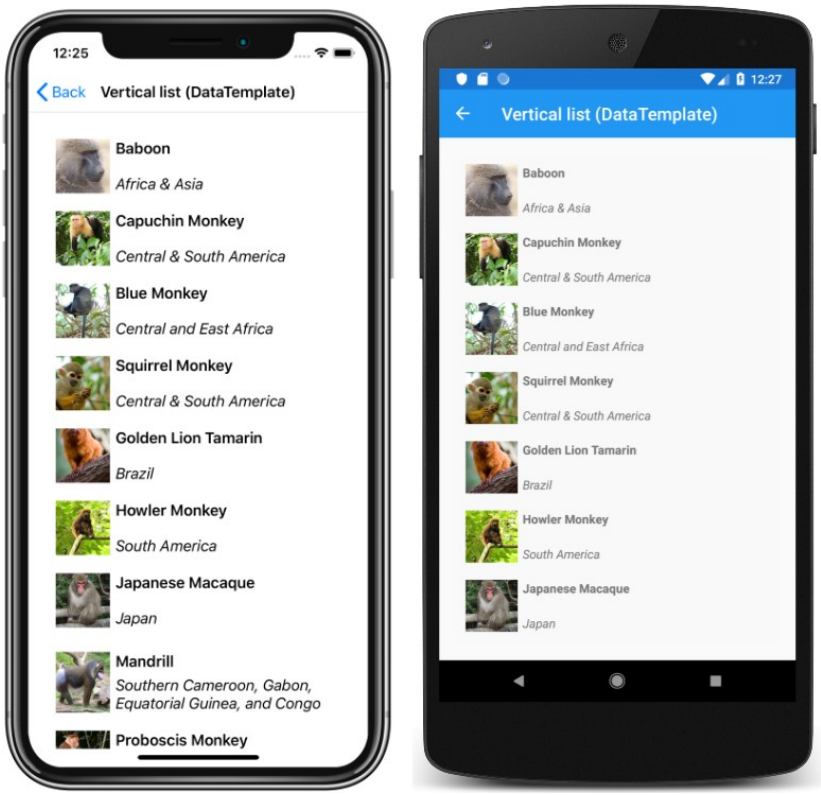
Příklad vytváří zobrazení pro obrázek a dva Label objekty. Obrázek bude o velikosti 2 řádků s definovanou výškou a šířkou na 60. Na obrázek jsou vázána data ImageUrl ze třídy Monkeys. Label objekty obsahují data Location a Name, také ze třídy Monkeys. Prvky Name a Location se zobrazí nad sebou, každý s jiným formátováním. Label Name bude tučným písmem, zatímco Location bude psaný kurzivou.

```
CollectionView collectionView = new CollectionView();
collectionView.SetBinding(ItemsView.ItemsSourceProperty, "Monkeys");

collectionView.ItemTemplate = new DataTemplate(() =>
{
    Grid grid = new Grid { Padding = 10 };
    grid.RowDefinitions.Add(new RowDefinition { Height = GridLength.Auto });
    grid.RowDefinitions.Add(new RowDefinition { Height = GridLength.Auto });
    grid.ColumnDefinitions.Add(new ColumnDefinition { Width = GridLength.Auto
});
    grid.ColumnDefinitions.Add(new ColumnDefinition { Width = GridLength.Auto
});

    Image image = new Image { Aspect = Aspect.AspectFill, HeightRequest = 60,
WidthRequest = 60 };
```

```
    image.SetBinding(Image.SourceProperty, "ImageUrl");

    Label nameLabel = new Label { FontAttributes = FontAttributes.Bold };
    nameLabel.SetBinding(Label.TextProperty, "Name");

    Label locationLabel = new Label { FontAttributes = FontAttributes.Italic,
VerticalOptions = LayoutOptions.End };
    locationLabel.SetBinding(Label.TextProperty, "Location");

    Grid.SetRowSpan(image, 2);

    grid.Children.Add(image);
    grid.Children.Add(nameLabel, 1, 0);
    grid.Children.Add(locationLabel, 1, 1);

    return grid;
});
```

Uvedený příklad kódu v pozadí pro nastavení vzhledu položky v seznamu znázorňuje, jakým způsobem lze spravovat rozložení položek v CollectionView pomocí Grid v jazyce C#. [26] Výsledné zobrazení viz obrázek 17.

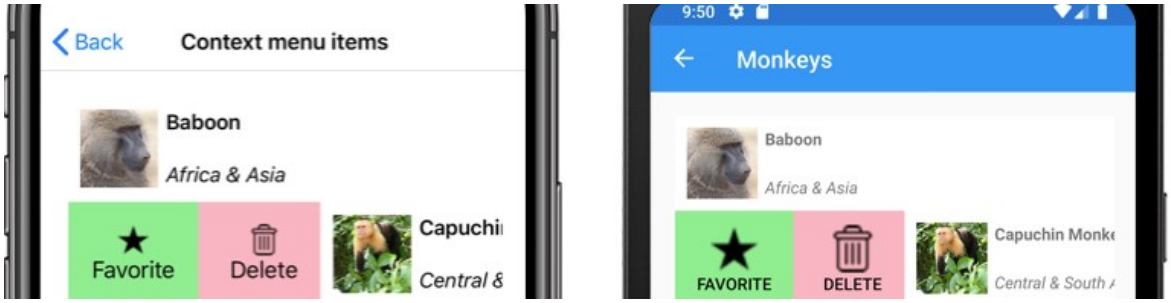
Grid uvedený v příkladu obsahuje 2 Label objekty, jeden pro jméno a druhý pro lokaci, a obrázek. Toto rozložení je vázáno na třídu „Monkeys“ ze které se čerpají data. Obrázek má nastavenou šířku a výšku na 60. Plocha s textem pro jméno má nastavený formát písma na tučné, zatímco plocha pro lokaci je formátována na text s kurzivou.



Obrázek 17 Výsledné zobrazení po nastavení vzhledu položek [26]

5.2.1 SwipeView

CollectionView podporuje další nabídku u položky (context menu) prostřednictvím SwipeView. Pomocí gesta potažení u položky se otevře kontextová nabídka viz obrázek 18. Využít tuto funkci lze tak, že do <ColelctionView.ItemTemplate> se zabalí <DataTemplate> a v <DataTemplate> se vyskytuje <SwipeView>. Ve SwipeView se definují položky, které mají být zobrazeny po gestu. Tyto položky se určují pomocí <SwipeItem>.



Obrázek 18 Využití SwipeView [26]

Tento nástroj je k dispozici pouze jako experimentální prvek. Pro využití tohoto nástroje je nutné, aby byl do aplikace přidán následující řádek kódu: „Forms.SetFlags("SwipeView_Experimental");“ před voláním Forms.Init.

5.2.2 Aktualizování zobrazení

CollectionView plně podporuje možnost aktualizaci zobrazení pomocí RefreshView. RefreshView poskytuje možnost aktualizovat data, která jsou zobrazena pouhým pohybem dolů po displeji. V tomto případě nastává změna a tou je zabalení CollectionView do RefreshView. Viz příklad používající XAML [26]:

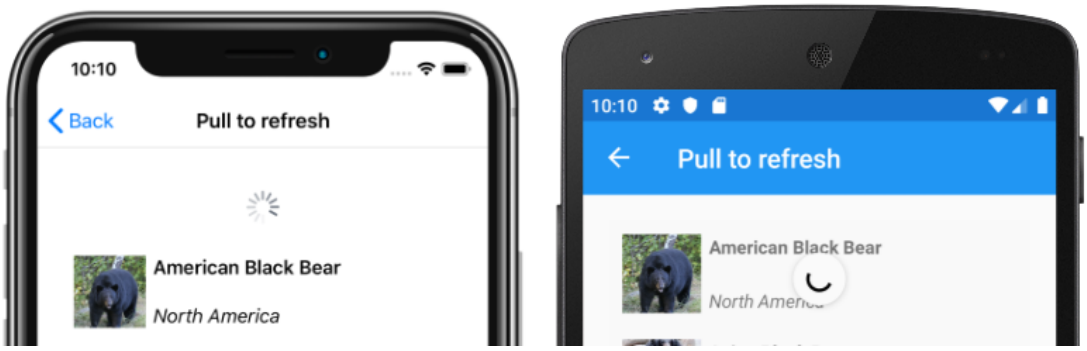
```
<RefreshView IsRefreshing="{Binding IsRefreshing}"
             Command="{Binding RefreshCommand}">
    <CollectionView ItemsSource="{Binding Animals}">
        ...
    </CollectionView>
</RefreshView>
```

Ekvivalentní kód v jazyce C# [26]:

```
RefreshView refreshView = new RefreshView();
ICommand refreshCommand = new Command(() =>
{
    // IsRefreshing is true
    // Refresh data here
    refreshView.IsRefreshing = false;
});
refreshView.Command = refreshCommand;

CollectionView collectionView = new CollectionView();
```

```
collectionView.SetBinding(ItemsView.ItemsSourceProperty, "Animals");
refreshView.Content = collectionView;
```

Jakmile uživatel chce aktualizovat položky v právě prohlíženém seznamu, stačí sjet prstem po displeji směrem dolů. Spustí se příkaz „RefreshCommand“ a objeví se na displeji kolečko, které signalizuje probíhanou aktualizaci dat v seznamu viz obrázek 19.



Obrázek 19 Aktualizace dat v seznamu [26]

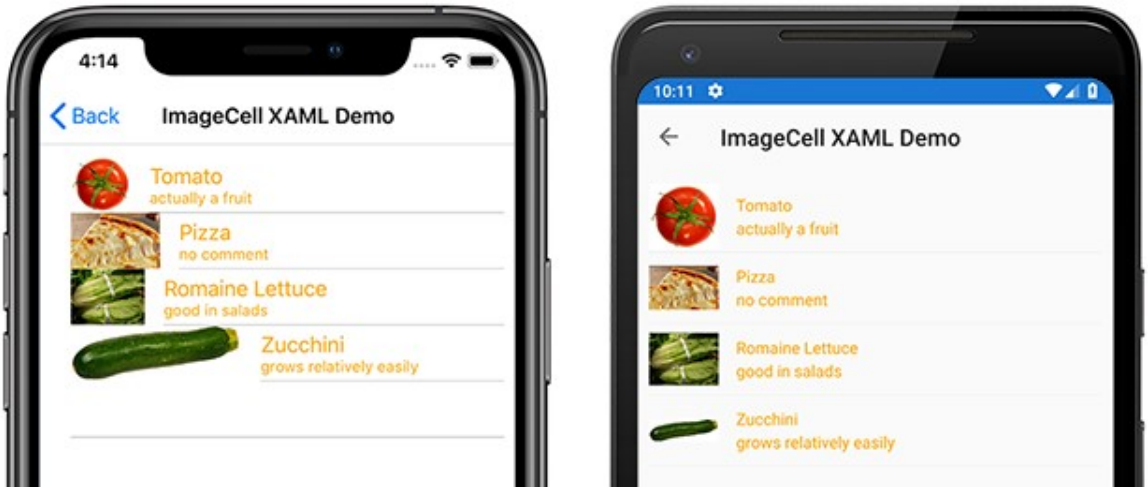
5.2.3 ListView

Dříve se používal a stále se používá ListView. CollectionView a ListView si jsou velmi podobné. CollectionView se používá pro prezentování dat, pro která je potřeba nastavit různé vzhledy, zároveň je tento způsob více flexibilní a výkonnější, než je ListView. Pomocí CollectionView lze jednoduchým způsobem udělat i horizontální zobrazení.

ListView se používá spíše pro obsáhlý list dat, mezi kterými je potřeba scrollovat. Zároveň obsahuje TextCell a ViewCell, kde TextCell slouží pouze k zobrazení textu, kdežto ViewCell se používá pro zobrazení určitým způsobem vzhledově upravených položek ze seznamu. Příklad kódu [27] pro použití ListView pro výsledné zobrazení viz obrázek 20:

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="demoListView.ImageCellPage">
    <ContentPage.Content>
        <ListView  x:Name="listView">
            <ListView.ItemTemplate>
                <DataTemplate>
                    <ViewCell>
                        <StackLayout BackgroundColor="#eee"
                        Orientation="Vertical">
                            <StackLayout Orientation="Horizontal">
                                <Image Source="{Binding image}" />
                                <Label Text="{Binding title}"
                                TextColor="#f35e20" />
                                <Label Text="{Binding subtitle}"
                                HorizontalOptions="EndAndExpand"
                                TextColor="#503026" />
```

```
                        </StackLayout>
                    </StackLayout>
                </ViewCell>
            </DataTemplate>
        </ListView.ItemTemplate>
    </ListView>
</ContentPage.Content>
</ContentPage>
```



Obrázek 20 Výsledné zobrazení pro ViewCell [27]

5.3 CarouselView

Je nástroj pro prezentování dat, kde se uživatel může jednoduchým způsobem pohybovat v seznamu položek. Zobrazuje horizontálně orientované položky, které se postupně zobrazují na displeji. Použitím gesta swipe – potažení probíhá přesun mezi položkami v seznamu.

CarouselView je implementací velice podobný nástoji CollectionView, avšak mají různé případy užití. CollectionView se spíše používá k zobrazení dat s libovolnou délkou. CarouselView je spíše využíván pro zvýraznění informací seznamu limitované délky.

Data do CarouselView jsou předávána pomocí vlastnosti ItemsSource, což je typ IEnumerable. IEnumerable určuje kolekci položek, jež mají být zobrazeny. Příklad bindování CarouselView daty pomocí jazyka XAML [29]:

```
<CarouselView ItemsSource="{Binding Monkeys}" />
```

Ekvivalentní kód v jazyce C#:

```
CarouselView carouselView = new CarouselView();
carouselView.SetBinding(ItemsView.ItemsSourceProperty, "Monkeys");
```

Příklad naplnění CarouselView daty pomocí XAML:

```
<CarouselView>
    <CarouselView.ItemsSource>
        <x:Array Type="{x:Type x:String}">
            <x:String>Baboon</x:String>
            <x:String>Capuchin Monkey</x:String>
            <x:String>Blue Monkey</x:String>
            <x:String>Squirrel Monkey</x:String>
            <x:String>Golden Lion Tamarin</x:String>
            <x:String>Howler Monkey</x:String>
            <x:String>Japanese Macaque</x:String>
        </x:Array>
    </CarouselView.ItemsSource>
</CarouselView>
```

Ekvivalentní kód v jazyce C# [29]:

```
CarouselView carouselView = new CarouselView();
carouselView.ItemsSource = new string[]
{
    "Baboon",
    "Capuchin Monkey",
    "Blue Monkey",
    "Squirrel Monkey",
    "Golden Lion Tamarin",
    "Howler Monkey",
    "Japanese Macaque"
};
```

Vzhled pro každou položku určuje šablona typu DataTemplate, která je určena vlastností ItemTemplate. Následuje příklad kódu v jazyce XAML, který nastaví rámeček položky na velikost 300, dále rámeček bude mít stín, zaoblení rohů a bude vycentrovaný. V rámečku bude textová plocha obsahující vycentrované jméno opice tučným písmem. Pod jménem bude v horizontální linii vycentrovaný obrázek o velikosti 150x150. Následují dvě pole s textem pro lokaci a krátký popis opice s omezením na maximálně 5 řádků psaný v kurzívě [29]:

```
<CarouselView ItemsSource="{Binding Monkeys}">
    <CarouselView.ItemTemplate>
        <DataTemplate>
            <StackLayout>
                <Frame HasShadow="True"
                       BorderColor="DarkGray"
                       CornerRadius="5"
                       Margin="20"
                       HeightRequest="300"
                       HorizontalOptions="Center"
                       VerticalOptions="CenterAndExpand">
                    <StackLayout>
                        <Label Text="{Binding Name}"
                               FontAttributes="Bold"
                               FontSize="Large"
                               HorizontalOptions="Center"
                               VerticalOptions="Center" />
```

```
                        <Image Source="{Binding ImageUrl}"
                               Aspect="AspectFill"
                               HeightRequest="150"
                               WidthRequest="150"
                               HorizontalOptions="Center" />
                        <Label Text="{Binding Location}"
                               HorizontalOptions="Center" />
                        <Label Text="{Binding Details}"
                               FontAttributes="Italic"
                               HorizontalOptions="Center"
                               MaxLines="5"
                               LineBreakMode="TailTruncation" />
                    </StackLayout>
                </Frame>
            </StackLayout>
        </DataTemplate>
    </CarouselView.ItemTemplate>
</CarouselView>
```

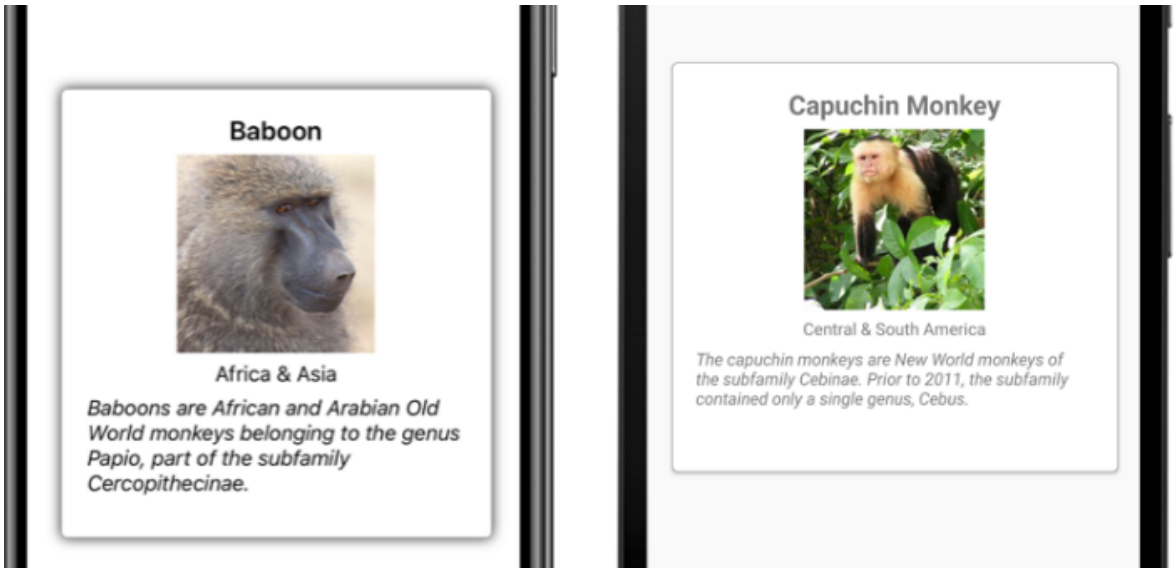
Ekvivalentní kód jazyka C# [29]:

```
CarouselView carouselView = new CarouselView();
carouselView.SetBinding(ItemsView.ItemsSourceProperty, "Monkeys");

carouselView.ItemTemplate = new DataTemplate(() =>
{
    Label nameLabel = new Label { ... };
    nameLabel.SetBinding(Label.TextProperty, "Name");

    Image image = new Image { ... };
    image.SetBinding(Image.SourceProperty, "ImageUrl");

    Label locationLabel = new Label { ... };
    locationLabel.SetBinding(Label.TextProperty, "Location");

    Label detailsLabel = new Label { ... };
    detailsLabel.SetBinding(Label.TextProperty, "Details");

    StackLayout stackLayout = new StackLayout
    {
        Children = { nameLabel, image, locationLabel, detailsLabel }
    };

    Frame frame = new Frame { ... };
    StackLayout rootStackLayout = new StackLayout
    {
        Children = { frame }
    };

    return rootStackLayout;
});
```

Výsledné zobrazení pro zmíněné kódy upravující vzhled položky viz obrázek 21.

Příklad třídy „Monkeys“ [29]:

```
public class Monkey
{
    public string Name { get; set; }
    public string Location { get; set; }
    public string Details { get; set; }
    public string ImageUrl { get; set; }
}
```



Obrázek 21 Výsledné zobrazení pomocí CarouselView [27]

CarouselView, stejně jako CollectionView podporuje SwipeView, kde je použití velmi podobné jako je tomu u CollectionView, a RefreshView, které se používá totožným způsobem, jako RefreshView u funkce CarouselView.

5.3.1 CarouselPage

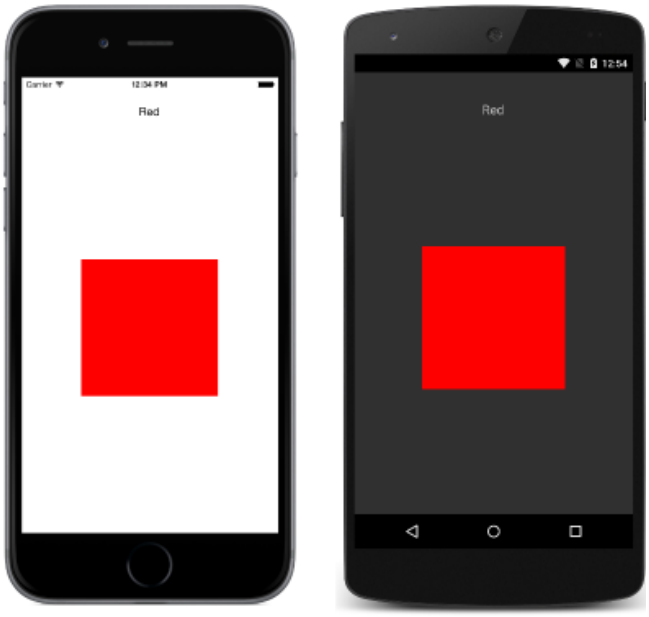
CarouselPage byla nahrazena CarouselView. Jsou používány dva způsoby, jimiž lze vytvořit CarouselPage. Prvním z nich je možnost naplnění CarouselPage pomocí ContentPage. V následujícím příkladu si lze všimnout funkce CarouselPage, který obsahuje tři ContentPage, kde první z ContentPages obsahuje textové pole s textem Red a uprostřed stránky se zobrazí červený čtverec, vše psané v jazyce XAML viz obrázek 22 [30]:

```
<CarouselPage xmlns="http://xamarin.com/schemas/2014/forms"
              xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
              x:Class="CarouselPageNavigation.MainPage">
    <ContentPage>
        <ContentPage.Padding>
          <OnPlatform x:TypeArguments="Thickness">
              <On Platform="iOS, Android" Value="0,40,0,0" />
          </OnPlatform>
        </ContentPage.Padding>
        <StackLayout>
```

```
            <Label Text="Red" FontSize="Medium" HorizontalOptions="Center" />
            <BoxView Color="Red" WidthRequest="200" HeightRequest="200"
HorizontalOptions="Center" VerticalOptions="CenterAndExpand" />
        </StackLayout>
    </ContentPage>
    <ContentPage>
        ...
    </ContentPage>
    <ContentPage>
        ...
    </ContentPage>
</CarouselPage>
```

Další možností je naplnění pomocí šablony, viz příklad [30]:

```
<CarouselPage xmlns="http://xamarin.com/schemas/2014/forms"
              xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
              x:Class="CarouselPageNavigation.MainPage">
    <CarouselPage.ItemTemplate>
        <DataTemplate>
            <ContentPage>
                <ContentPage.Padding>
                  <OnPlatform x:TypeArguments="Thickness">
                    <On Platform="iOS, Android" Value="0,40,0,0" />
                  </OnPlatform>
                </ContentPage.Padding>
                <StackLayout>
                    <Label Text="{Binding Name}" FontSize="Medium"
HorizontalOptions="Center" />
                    <BoxView Color="{Binding Color}" WidthRequest="200"
HeightRequest="200" HorizontalOptions="Center"
VerticalOptions="CenterAndExpand" />
                </StackLayout>
            </ContentPage>
        </DataTemplate>
    </CarouselPage.ItemTemplate>
</CarouselPage>
```



Obrázek 22 Výsledné zobrazení první stránky příkladných kódů.[30]

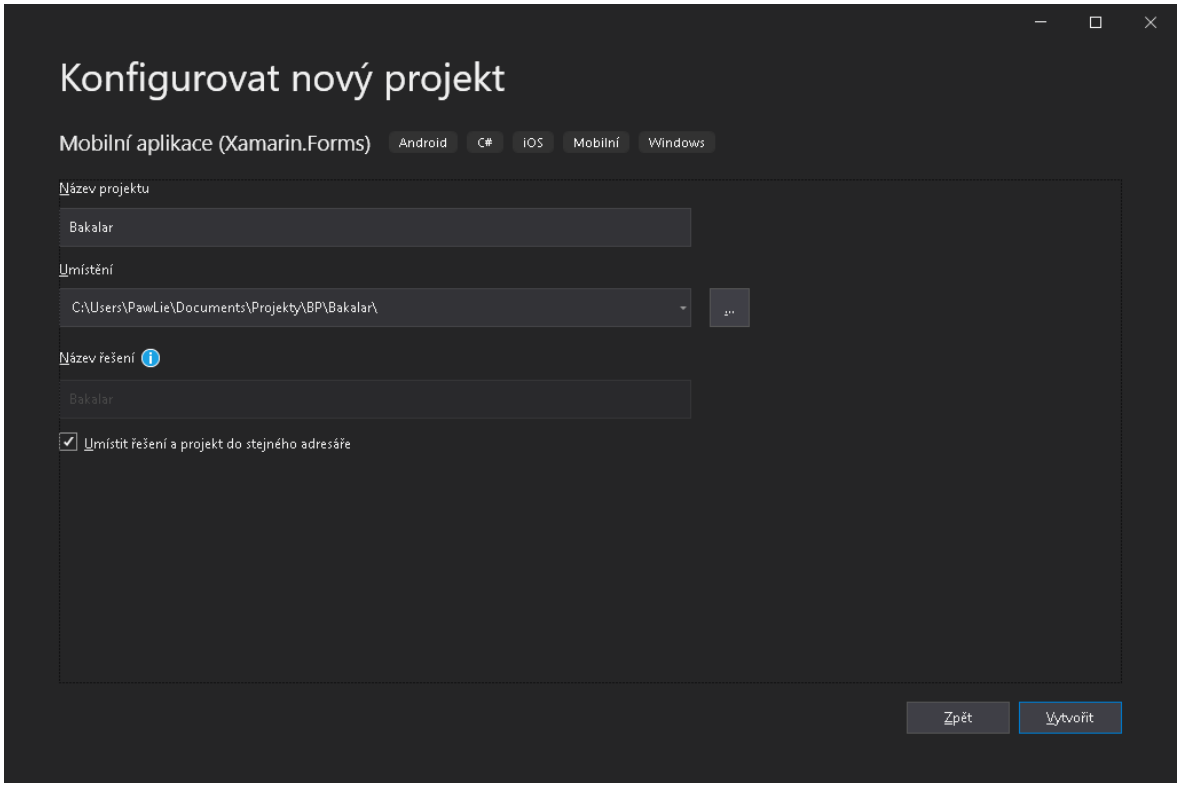
II. PRAKTICKÁ ČÁST

6 APLIKACE DEMONSTRUJÍCÍ NOVÉ FUNKCE XAMARIN.FORMS

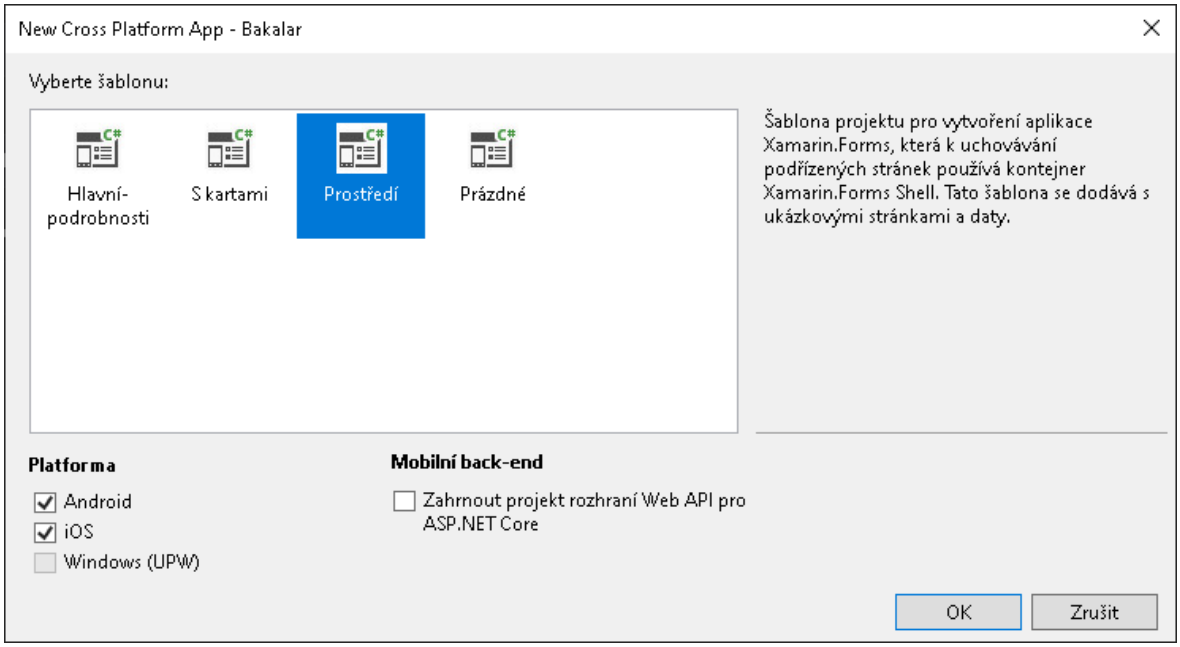
Praktickou část této práce tvoří multiplatformně vyvíjená aplikace využívajíc framework Xamarin.Forms. Aplikace slouží k demonstraci využití nových funkcí, kterými jsou CarouselView, Shell a CollectionView.

6.1 Založení projektu

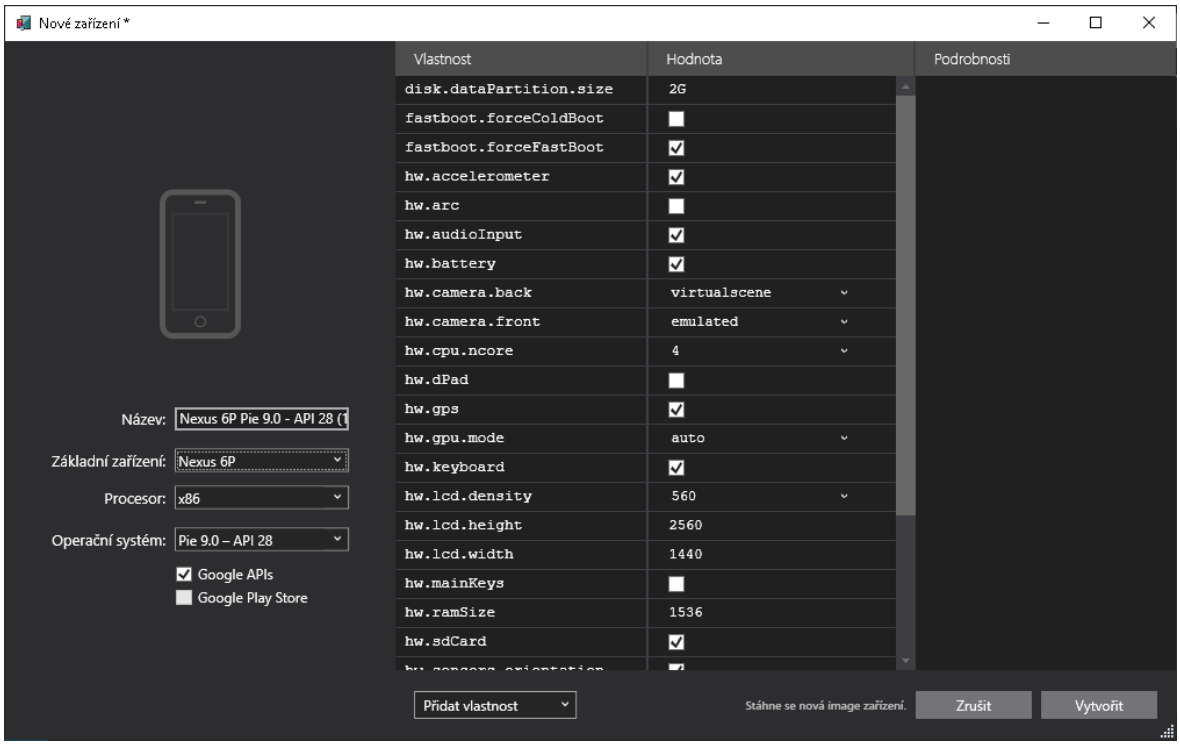
Pro možnost použití všech funkcí, které budou v následujících kapitolách zmíněny bylo zapotřebí vytvořit Xamarin.Forms projekt s využitím šablony Shell. viz obrázky 23, 24. Veškeré funkce aplikace byly vyzkoušeny na emulovaném zařízení Nexus 6P s operačním systémem Android Pie 9.0 viz obrázek 25



Obrázek 23 Vytvoření projektu s frameworkem Xamarin.Forms



Obrázek 24 Vytvoření projektu s použitím šablony Shell



Obrázek 25 Vytvoření emulátoru-zařízení Nexus 6P

7 TVORBA DAT PRO ZOBRAZENÍ

Aby bylo možné správně demonstrovat nové funkce frameworku Xamarin.Forms, je nutné si vytvořit data, kterými se budou vytvořené stránky aplikace plnit. Vybral jsem si, že moje data budou tvořena plemeny psů.

7.1 Třída Dog.cs

Prvotním cílem je vytvořit třídu, která obsahuje definice pro každé plemeno. V následujícím kódu jsou proměnné, které obsahují dané definice:

```
public class Dog
    {
        string descripS;
        string descrip;
        public int ID { get; set; }
        public string Name { get; set; }
        public string ImageUrl { get; set; }
        public string Favourite { get; set; }
        public string ShortDescription {
            get {
                if (descripS == null)
                {
                    descripS = "Pes " + Name;
                }
                return descripS;
            }
            set {
                descripS = value;
            }
        }
        public string Description {
            get
            {
                if (descrip == null)
                {
                    descrip = "Dlouhý popisek pro psa " + Name;
                }
                return descrip;
            }
            set {
                descrip = value;
            }
        }
    }
```

Tato třída obsahuje definice pro ID, Název plemene, string pro adresu obrázku, zda-li je plemeno oblíbené, nebo není, krátký a dlouhý popis. Kvůli jednodušší práci se zobrazením je zvoleno způsob využití string pro proměnnou Favourite namísto bool. Dále třída obsahuje dvě pomocné proměnné descrip a descripS, které slouží pro automatické generování popisku plemene. Krátký popis plemene je generován pouze tak, že se do této proměnné zapíše

textový řetězec „Pes“ a k tomu se přidá název plemene. Dlouhý popisek funguje na stejném principu, jen se do textového řetězce přidá "Dlouhý popisek pro psa " a název plemene.

7.2 Tvorba dat

Po vytvoření třídy Dog je nutné vytvořit nějaká data pro budoucí demonstraci funkcí zobrazení. Tvorba dat je prováděna v konstruktoru třídy MockDataStore, které se i mimo jiné stará o práci s daty, jako je přidávání plemene, odstranění plemene. Následující kód byl použit k naplnění listu dat plemeny psů. Informace a odkazy na fotky byly čerpány z [31].

```
    public MockDataStore()
        {
            dogs = new List<Dog>() {
                new Dog()
                {
                    Name = "Akita",
                    ImageUrl =
"https://www.spokojenypes.cz/ImgP.ashx?co=/ImgGalery/Img1/Atlas/170308111919-
akita2-jpg.jpg&fd=a5&pa=1"
                },
                new Dog()
                {
                    Name = "Německý ovčák",
                    ImageUrl =
"https://www.spokojenypes.cz/ImgP.ashx?co=/ImgGalery/Img1/Atlas/180827103124-
nemecky-ovcak-cissi-karel-jpg.jpg&fd=a5&pa=1"
                },
                new Dog()
                {
                    Name = "Australský ovčák",
                    ImageUrl =
"https://www.spokojenypes.cz/ImgP.ashx?co=/ImgGalery/Img1/Atlas/170308112046-
australsky-ovcak-jpg.jpg&fd=a5&pa=1"
                },
                ...
            };
            for (int i = 0; i < dogs.Count; i++)
            {
                dogs[i].ID = i;
                dogs[i].Favourite = "Ne";
            }
        }
```

Pomocí uvedeného kódu byla vytvořena data sloužící k demonstraci zobrazení. Byl vytvořen list dat třídy Dog, do kterého byla vložena plemena se jmény Akita, Německý ovčák a Australský ovčák. Po vytvoření listu se celý list projde pomocí cyklu a doplní se ID každému plemeni. Zároveň je pro každé plemeno nastaven text „Ne“ do proměnné oblíbený.

7.3 Práce s daty

V následující kapitole jsou popsány některé z metod, které pracují s daty. Jsou to jednoduché metody provádějící základní práci s daty.

7.3.1 Metoda pro přidání plemene

```
public async Task<bool> AddDogAsync(Dog dog)
        {
            dogs.Add(dog);
            return await Task.FromResult(true);
        }
```

Zmíněný kód slouží k přidání plemene psa. V parametru přijme plemeno psa a následně ho přidá do listu plemen s názvem dogs.

7.3.2 Metoda pro smazání plemene

```
        public async Task<bool> DeleteDogAsync(Dog dog)
        {
            var oldItem = dogs.Where((Dog arg) => arg.ID ==
dog.ID).FirstOrDefault();
            dogs.Remove(oldItem);

            return await Task.FromResult(true);
        }
```

Uvedená metoda slouží k odstranění plemene. V parametru přijme plemeno psa a následně ho přidá do listu plemen s názvem dogs.

7.3.3 Metoda pro změnu nastavení proměnné oblíbený

Následující kód slouží ke změně v proměnné oblíbenosti u plemene. V parametru přijme plemeno, vytvoří se string result, který obsahuje text „Ne“ a následně kontroluje, jestli má dané plemeno v proměnné Favourite hodnotu „Ne“. Pokud obsahuje danou hodnotu, hodnota proměnné Favourite se přepíše na „Ano“ a to je následně uloženo do proměnné Favourite u daného plemena. Pokud proměnná Favourite obsahuje „Ano“ nenastává v result žádná změna a ukládá se původní hodnota, tedy „Ne“ do proměnné Favourite.

```
  public async Task<bool> FavouriteDogAsync(Dog dog)
        {
            string result = "Ne";
            if (dog.Favourite.Equals("Ne"))
            {
                result="Ano";
            }
            dog.Favourite = result;

            return await Task.FromResult(true);
        }
```

7.4 Volání metod

Některá zobrazení, jako například zobrazení demonstrující použití funkce CollectionView, či zobrazení pro detail plemene, budou obsahovat tlačítka spouštějíc metody uvedené v kapitole 7.3. Tyto tlačítka pro svoji správnou funkčnost potřebují kód v pozadí aplikace, který umožní přístup k daným metodám.

Stránka CollectionPage.xaml, která slouží jako zobrazení pro využití funkce CollectionView obsahuje prvek SwipeView. Tento prvek obsahuje dvě tlačítka viz obrázek 29:

```
<SwipeItems>
             <SwipeItem Text="Oblíbené"
              BackgroundColor="BlueViolet"
              Invoked="FavouriteChange"/>
             <SwipeItem Text="Detaily"
              BackgroundColor="Gray"
               Invoked="OnItemSelected"/>
</SwipeItems>
```

Každé z těchto tlačítek obsahuje vlastnosti definující jeho barvu, text, který se zobrazí a metodu, která se po stisknutí tlačítka ve třídě CollectionPage.xaml.cs spustí.

Následující kód je využit k zobrazení stránky pro detail plemene. Plemeno je předáno stránce DogDetailPage jako parametr a vytvoří se zobrazení viz obrázek 30.

```
async void OnItemSelected(object sender, EventArgs args)
        {
            var layout = (BindableObject)sender;
            var dog = (Dog)layout.BindingContext;
            await Navigation.PushModalAsync(new DogDetailPage(new
DogDetailViewModel(dog)));
        }
```

Druhé tlačítko SwipeView má za úkol změnit hodnotu v proměnné favourite.

```
void FavouriteChange(object sender, EventArgs e)
        {
            var layout = (BindableObject)sender;
            var dog = (Dog)layout.BindingContext;
            MessagingCenter.Send(this, "FavouriteDog", dog);
        }
```

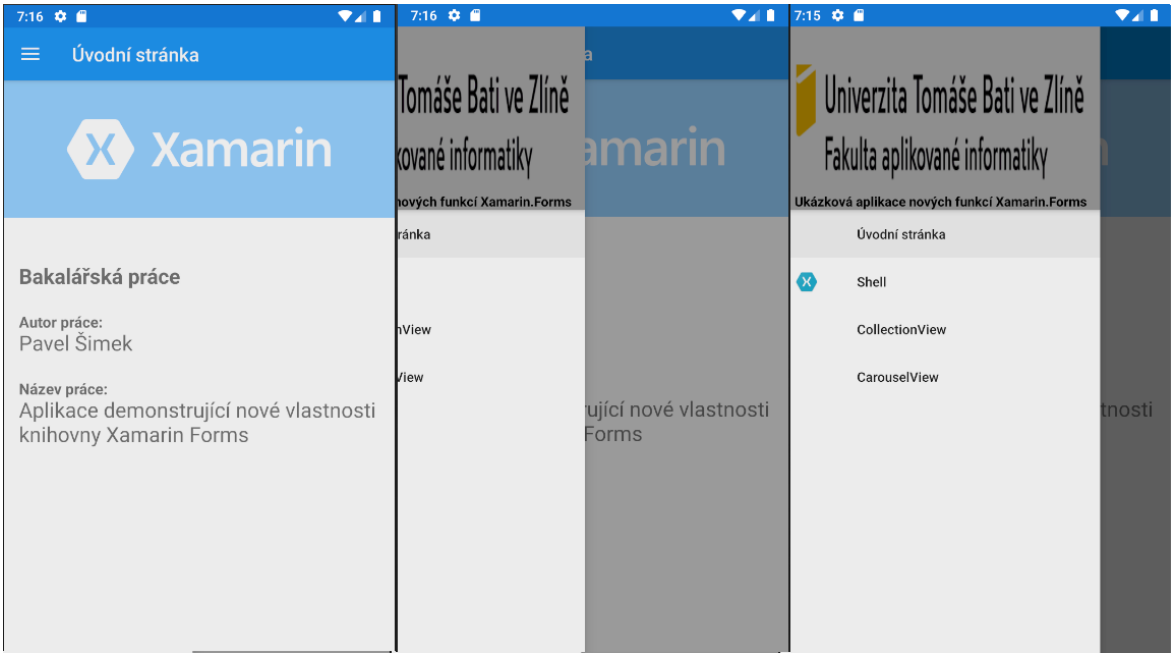
Toho je dosaženo tím způsobem, že se odešle do třídy CollectionViewModel.cs zpráva, že má být zahájena metoda obsahující název FavouriteDog, což spustí níže uvedený kód:

```
MessagingCenter.Subscribe<CollectionPage, Dog>(this, "FavouriteDog", async
(obj, dog) =>
            {
                await DataStore.FavouriteDogAsync(dog);
            });
```

Tento kód vykoná finální asynchronní metodu uvedenou v kapitole 7.3.3, která změní hodnotu proměnné favourite u daného plemena.

8 SHELL

Základem aplikace je využití funkcionality Shell. Použitím Shell bylo vytvořeno menu aplikace, které umožnuje pohyb v aplikaci. Na úvodní stránce je možnost navigace v aplikaci pomocí FlyoutItem. Po kliknutí na tři rovnoběžné čáry v levém horním rohu obrazovky, nebo přetažením po displeji z leva doprava, se zobrazí postranní menu, díky kterému je možné se v aplikaci pohybovat na jiná zobrazení viz obrázek 26.



Obrázek 26 Zobrazení Flyout menu

Na obrázku 26 lze mimo jiné vidět, logo Univerzity Tomáš Bati a popisek. Tato část menu byla vytvořena pomocí kódu:

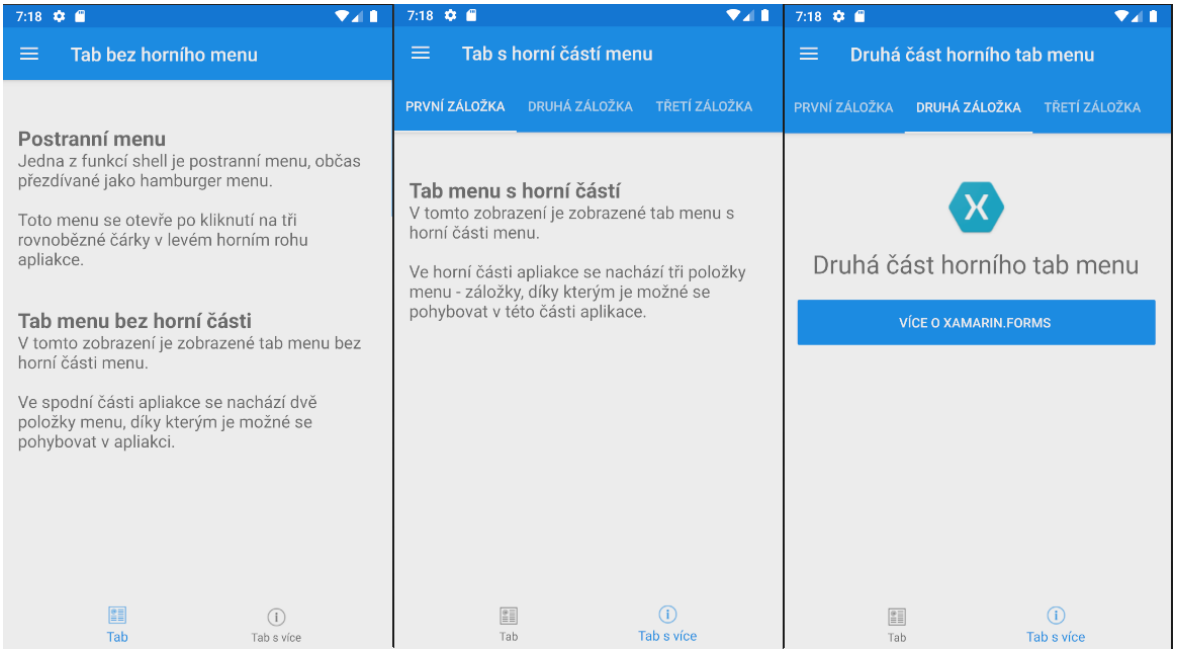
```
<Shell.FlyoutHeaderTemplate>
        <DataTemplate>
            <Grid BackgroundColor="Black">
                <Image Aspect="Fill"
                   Source="fai.jpg"
                   Opacity="0.6" />

            <Label Text="Ukázková aplikace nových funkcí Xamarin.Forms"
                   HeightRequest="100"
                   TextColor="Black"
                   FontAttributes="Bold"
                   HorizontalTextAlignment="Center"
                   VerticalTextAlignment="End" />
             </Grid>
        </DataTemplate>
    </Shell.FlyoutHeaderTemplate>
```

Obrázek má nastavený zdroj „fai.jpg“, viditelnost a způsob, jakým se má zobrazit, v tomto případě vyplnit daný prostor. Následující položky menu jsou tvořeny pomocí FlyoutItem.

Pro ukázku možností menu v Shell bylo vytvořeno zobrazení s využitím Tab menu:

```
<FlyoutItem Title="Úvodní stránka">
        <ShellContent ContentTemplate="{DataTemplate views:ShellPage}" />
    </FlyoutItem>
    <FlyoutItem Title="Shell" Icon="icon.png">
        <Tab Title="Tab" Icon="tab_feed.png">
            <ShellContent ContentTemplate="{DataTemplate views:TabPage}"/>
        </Tab>
        <Tab Title="Tab s více" Icon="tab_about.png">
            <ShellContent Title="První záložka" ContentTemplate="{DataTemplate
views:MultiTabPage1}" />
            <ShellContent Title="Druhá záložka" ContentTemplate="{DataTemplate
views:MultiTabPage2}" />
            <ShellContent Title="Třetí záložka" ContentTemplate="{DataTemplate
views:MultiTabPage3}" />
        </Tab>
    </FlyoutItem>
    <FlyoutItem Title="CollectionView" >
        <ShellContent ContentTemplate="{DataTemplate views:CollectionPage}" />
    </FlyoutItem>
    <FlyoutItem Title="CarouselView">
        <ShellContent ContentTemplate="{DataTemplate views:CarouselPage}"/>
    </FlyoutItem>
```

V uvedeném kódu je ukázáno, co se v daných položkách menu zobrazí. První z FlyoutItem zobrazí úvodní stránku, v aplikaci pojmenovanou ShellPage. Druhý FlyoutItem poskytne zobrazení se dvěmi položkami menu ve spodní části obrazovky. Tyto položky jsou definované pomocí Tab uzavřené ve FlyoutItem. Druhá z těchto položek obsahuje ShellContent, díky čemuž se v daném zobrazení zobrazí tři záložky v horní části aplikace viz obrázek 27. Vlastnost Title ve FlyoutItem, Tab a ShellContent slouží pro pojmenování položky. Vlastnost Icon k zobrazení ikonky pro položku menu.

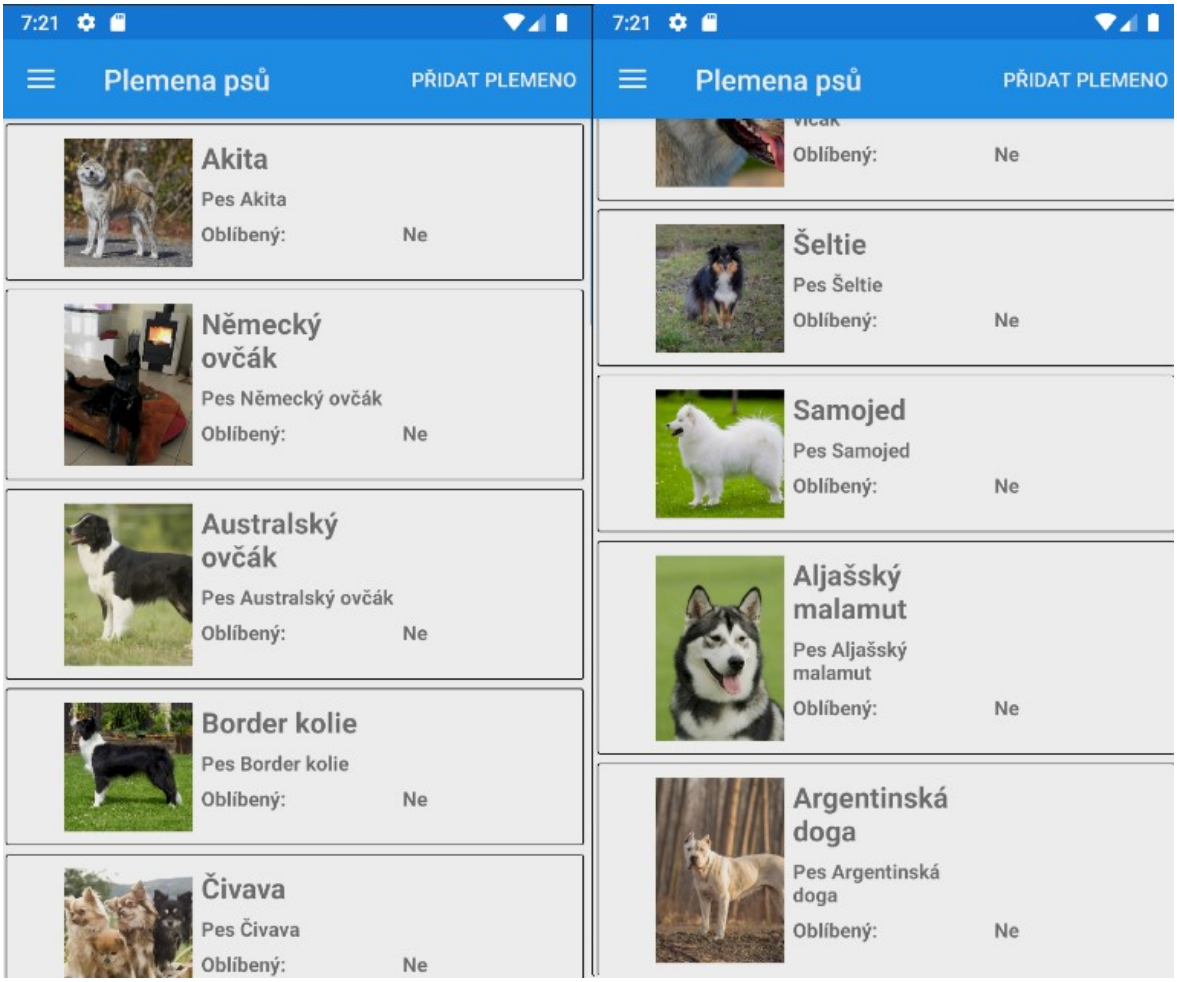


Obrázek 27 Průchod TabMenu

Následují další dvě položky v postranním menu a těmi jsou CollectionView a CarouselView. Jak tyto názvy napovídají, dané položky obsahují zobrazení s využitím funkcí CollectionView a CarouselView.

9 COLLECTIONVIEW

CollectionView poskytuje zobrazení pro data. Lze jednoduchým způsobem upravovat vzhled zobrazení položek na seznamu. K vytvoření zobrazení viz obrázek 28 je potřeba vytvořit třídu, která obsahuje daná data o plemeni psa a některá další data, pro demonstraci funkcí, které jsou následně využity.



Obrázek 28 Možnost scrollování mezi daty

9.1 Vzhled zobrazení

V následujícím kódu je uveden kód, který se stará o vzhled položek:

```
<Frame BorderColor="Black" Padding="5">
                                <Grid Padding="5">
                                    <Grid.RowDefinitions>
                                        <RowDefinition Height="Auto" />
                                        <RowDefinition Height="Auto" />
                                        <RowDefinition Height="Auto" />
                                    </Grid.RowDefinitions>
                                    <Grid.ColumnDefinitions>
                                        <ColumnDefinition Width="25" />
                                        <ColumnDefinition Width="Auto" />
                                        <ColumnDefinition Width="135" />
                                    </Grid.ColumnDefinitions>
                                    <Image Grid.RowSpan="3"
                                       Grid.Column="1"
                                   Source="{Binding ImageUrl}"
                                   Aspect="AspectFill"
                                   HeightRequest="90"
                                    WidthRequest="90"
                                   HorizontalOptions="CenterAndExpand" />
                                    <Label Grid.Column="2"
                                   Text="{Binding Name}"
                                   FontAttributes="Bold"
                                           FontSize="20"/>
                                    <Label Grid.Row="1"
                                   Grid.Column="2"
                                   Text="{Binding ShortDescription}"
                                   FontAttributes="Bold" />
                                    <Label Grid.Row="2"
                                   Grid.Column="2"
                                   Text="Oblíbený: "
                                   FontAttributes="Bold" />
                                    <Label Grid.Row="2"
                                   Grid.Column="3"
                                   Text="{Binding Favourite}"
                                   FontAttributes="Bold" />
                                </Grid>
                            </Frame>
```

Funkce Frame nastaví rámeček položky s černou barvou a odsazením. Dále je použita mřížka, tedy Grid, jejíž funkce je vytvořit plochu pro uspořádání jednotlivých prvků v kartě plemena psa. V mřížce jsou umístěny obrázek – Image, a čtyři textová pole. Pro obrázek je nastavena velikost 90x90, horizontální centrování s roztažením a zdroj na ImageUrl z třídy „Dog.cs“. Grid.Column a Grid.Row slouží k umístění daných prvků na mřížce. Pro textová pole se zdroj dat používá také ze třídy Dog.cs. Nastavení těchto polí může být různé, zde je použito tučného písma a pro název plemene psa je písmo zvětšeno na velikost 20.

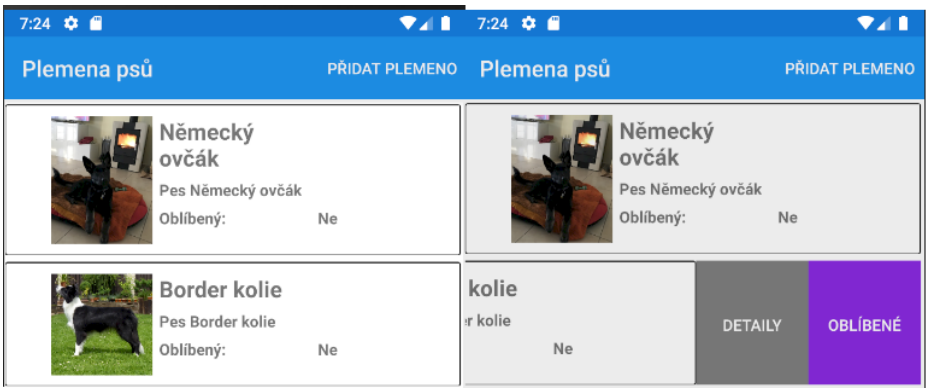
Generovaná data se načítají pomocí: `<CollectionView ItemsSource="{Binding Dogs}">`

Do horní části tohoto zobrazení je přidané tlačítko pro přidání plemena psa. Kód pro zobrazení tlačítka:

```
<ContentPage.ToolbarItems>
        <ToolbarItem Text="Přidat plemeno" Clicked="AddDog_Clicked" />
    </ContentPage.ToolbarItems>
```

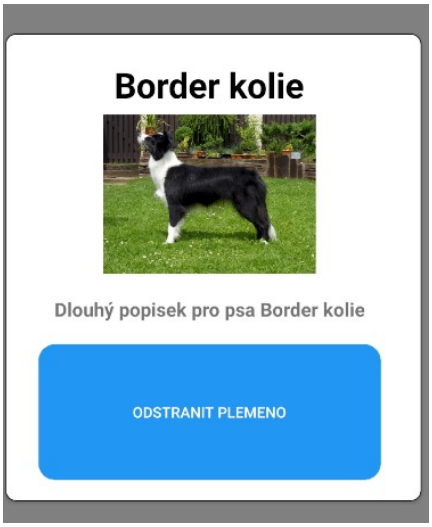
9.2 SwipeView

SwipeView poskytuje možnost přidat každé položce „neviditelné možnosti“. Neviditelné z důvodu, že se tyto možnosti zobrazí pouze po vykonání určitého gesta, v případě této aplikace je to gesto přejetí zprava doleva viz obrázek 29.



Obrázek 29 Zobrazení SwipeView

Tlačítko Detaily slouží k zobrazení podrobností o daném plemeni viz obrázek 30



Obrázek 30 Detailní stránka otevřená pomocí nabídky SwipeView

Tlačítko oblíbené umožní uživateli dát plemeni psa status oblíbeného plemena. V tomto případě se ovšem změna neprojeví ihned a je důležité celé zobrazení aktualizovat, k čemuž slouží RefreshView ukázané v následující kapitole 8.3.

Aby bylo možné SwipeView využít, je nutné přidat následující kód do základní třídy každé platformy: `Xamarin.Forms.Forms.SetFlags("SwipeView_Experimental");`

Důvodem je, že SwipeView je experimentální funkce a není uvedena jako funkce stabilní.

9.3 RefreshView

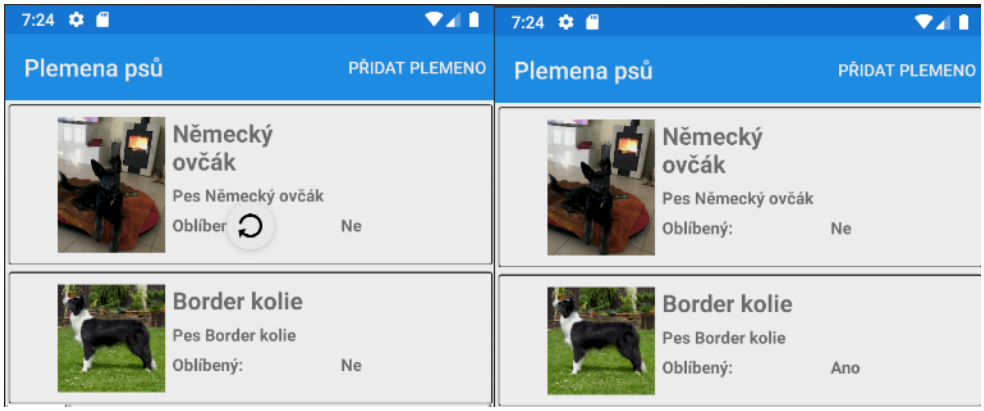
Do CollectionView je možné přidat funkcionalitu RefreshView, její cílem je aktualizování zobrazení, když chce sám uživatel. Využitím následujícího kódu je možné po gestu sjetí prstem po displeji směrem dolů aktualizovat data v zobrazení:

```
<RefreshView IsRefreshing="{Binding IsBusy, Mode=TwoWay}" Command="{Binding LoadItemsCommand}">
```

Uvedený kód způsobí, že se v kódu na pozadí zavolá příkaz, který vede k aktualizování zobrazení. Příklad kódu:

```
LoadItemsCommand = new Command(async () => await ExecuteLoadItemsCommand());

  public async Task ExecuteLoadItemsCommand()
        {
            IsBusy = true;

            try
            {
                Dogs.Clear();
                var dogs = DataStore.GetDogs();
                foreach (var dog in dogs)
                {
                    Dogs.Add(dog);
                }
            }
            catch (Exception ex)
            {
                Debug.WriteLine(ex);
            }
            finally
            {
                IsBusy = false;
            }
        }
```

Po provedení gesta se objeví šipka směřující dokola, což značí, že probíhá aktualizování zobrazení. Na obrázku 31 lze vidět, jak daná šipka vypadá a u položky oblíbené v kartě plemena Border kolie je možné pozorovat změnu z Ne na Ano.



Obrázek 31 Využití nástroje RefreshView

10 CAROUSELVIEW

Pomocí CarouselView bylo vytvořeno zobrazení jednotlivých dat, mezi kterými se lze pohybovat pomocí gesta posunutí prstem, swipe.

10.1 Vzhled CarouselView

V CarouselView bylo opět použito funkce RefreshView, která má stejnou funkci, jako tomu bylo v CollectionView. Příklad:

```
<RefreshView IsRefreshing="{Binding IsBusy, Mode=TwoWay}" Command="{Binding
LoadItemsCommand}">

 </RefreshView>
```

Obalené v RefreshView se následně nachází CarouselView s vlastností pro data, neboli ItemsSource, viz příklad:

```
<CarouselView ItemsSource="{Binding Dogs}">

<CarouselView>
```

V CarouselView se definuje vzhled položek v daném zobrazení. Výsledné zobrazení i s posuvem mezi položkami lze vidět na obrázku 31. Příklad kódu s nastavující způsob, jakým budou položky zobrazeny:

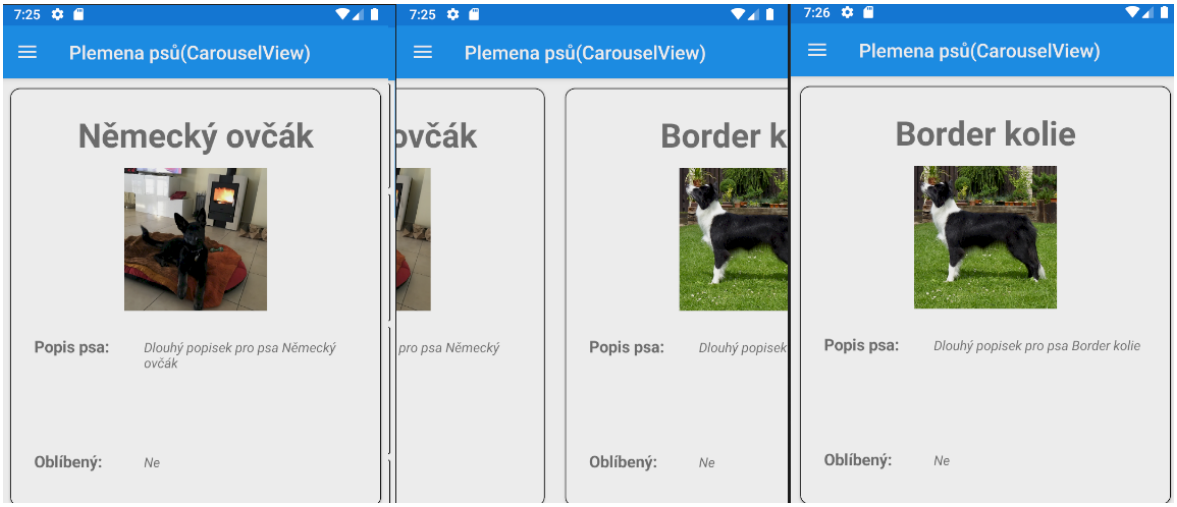
```
<CarouselView.ItemTemplate>
                    <DataTemplate>
                    <StackLayout>
                      <Frame Margin="10" CornerRadius="10" BorderColor="Black"
HeightRequest="400" WidthRequest="350" HorizontalOptions="Center"
VerticalOptions="Center" HasShadow="True">
                        <Grid Padding="5">
                           <Grid.RowDefinitions>
                              <RowDefinition Height="Auto" />
                              <RowDefinition Height="Auto" />
                              <RowDefinition Height="Auto" />
                          </Grid.RowDefinitions>
                          <Label Grid.Row="0" Grid.ColumnSpan="3"
                           HorizontalOptions="Center" VerticalOptions="Center"
                             Text="{Binding Name}"
                                   FontAttributes="Bold" FontSize="35" />
                              <Image Grid.Row="2" Grid.ColumnSpan="3"
                                 Source="{Binding ImageUrl}"
                                 Aspect="AspectFill"
                                 HeightRequest="150"
                                 WidthRequest="150"
                                     VerticalOptions="CenterAndExpand"
                                     HorizontalOptions="CenterAndExpand"/>
                                <Label Grid.Row="3"
                                       Grid.RowSpan="2"
                                       Grid.Column="0"
                                       Margin="0, 20, 15, 5"
                                 Text="Popis psa: "
```

```
                        FontAttributes="Bold"
                                FontSize="17"/>
                         <Label Grid.Row="3"
                                Grid.Column="1"
                                Grid.RowSpan="2"
                                Grid.ColumnSpan="2"
                                Margin="0,23, 0, 0"
                        Text="{Binding Description}"
                        FontAttributes="Italic" />

                         <Label Grid.Row="5"
                                Grid.Column="0"
                                Margin="0, 20, 15, 5"
                        Text="Oblíbený: "
                        FontAttributes="Bold"
                                FontSize="17"/>
                         <Label Grid.Row="5" Grid.Column="1"
                          Grid.ColumnSpan="2" Margin="0,23, 0, 0"
                        Text="{Binding Favourite}"
                        FontAttributes="Italic" />
                    </Grid>
                </Frame>

            </StackLayout>
        </DataTemplate>
    </CarouselView.ItemTemplate>
```

Kód definuje černý rámeček se zaoblenými rohy o velikosti 350x400. Rámeček bude vycentrovaný v horizontálním i vertikálním směru a bude mít nastavený stín. Do rámečku je umístěna mřížka, do které se vkládají jednotlivé položky, které mají být zobrazeny.

Jako první položka je zobrazeno textové pole, které je opět horizontálně i vertikálně vycentrované a obsahuje text s názvem plemene. Pod prvním textovým polem se nachází obrázek plemena, který obsahuje pouze odkaz na obrázek. Aplikace se sama stará o to, aby byl obrázek zobrazen. Obrázek plemene psa je také vycentrovaný.

Následují čtyři textová pole umístěna způsobem, že první dvě textová pole ze zmíněných čtyř jsou vedle sebe, kde první obsahuje statický text „Popis psa:“ a druhý se generuje při vytváření zobrazení. Následující dvě textová pole jsou ve spodní části rámečku, kde jedno obsahuje opět statický text: „Oblíbený: “ a druhé pole obsahuje text, zda-li pes obsahuje Ano, nebo Ne, tento text se také generuje při vytváření zobrazení.



Obrázek 32 Využití nástroje CarouselView s gestem posunutí

V uvedeném kódu lze vidět prostor mezi </Frame> a </StackLayout> ve kterém je v aplikaci níže uvedený kód:

```
<StackLayout.GestureRecognizers>
       <TapGestureRecognizer NumberOfTapsRequired="1"
       Tapped="OnItemSelected"></TapGestureRecognizer>
</StackLayout.GestureRecognizers>
```

Tento kód pomocí kódu na pozadí aplikace spustí po kliknutí na displej zobrazení karty plemene viz obrázek 30. Příklad kódu na pozadí aplikace:

```
     async void OnItemSelected(object sender, EventArgs args)
        {
            var layout = (BindableObject)sender;
            var dog = (Dog)layout.BindingContext;
            await Navigation.PushModalAsync(new DogDetailPage(new
DogDetailViewModel(dog)));
        }
    }
```

Spustí se asynchronní proces, který postupně zjistí, jaké plemenu bylo právě zobrazeno, získá se instance tohoto plemene a tato instance se jako parametr odešle do DogDetailPage.

10.1.1 DogDetailPage

V DogDetailPage se nachází následující kód, který vytvoří zobrazení viz obrázek 30:

```
  <StackLayout HorizontalOptions="Center" VerticalOptions="Center">
        <Frame Margin="10"
           CornerRadius="10"
           BorderColor="Black"
           HeightRequest="400"
           WidthRequest="400"
           HorizontalOptions="Center"
           VerticalOptions="Center"
               HasShadow="True">
            <Grid HorizontalOptions="Center" >
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto"  />
                </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition Width="Auto" />
            </Grid.ColumnDefinitions>
                <Label Grid.Row="1"
                                    Text="{Binding Dog.Name}"
                                    FontAttributes="Bold"
HorizontalOptions="Center" FontSize="33" TextColor="Black"
MinimumHeightRequest="60"/>
                <Image Grid.Row="2" Source="{Binding Dog.ImageUrl}"
                                    Aspect="AspectFit"
                                    MinimumHeightRequest="150" />
                <Label Grid.Row="3" Margin="15"
                                    FontSize="18"
                                    Text="{Binding Dog.Description}"
                                    FontAttributes="Bold"
MinimumHeightRequest="100"/>
                <Button Text="Odstranit plemeno" Grid.Row="4" CornerRadius="15"
MinimumHeightRequest="65"
                        BackgroundColor="#2196F3"
                        Clicked="RemoveDog_Clicked"
                        TextColor="White" />

        </Grid>

        </Frame>
    </StackLayout>
```

Vzhled zobrazované položky pomocí této stránky je velmi podobný jedné položce z CarouselView. Opět je zde černý rámeček se zaoblenými rohy. V rámečku se nachází vycentrovaný název plemene, pod ním obrázek a následuje textové pole pro dlouhý popis plemene. Pod popisem je umístěno tlačítko odstranit plemeno. Toto tlačítko pomocí `Clicked="RemoveDog_Clicked"` zavolá kód v pozadí, který plemenu z listu položek odstraní.

Kód v pozadí starající se o odstranění plemene:

```
async void RemoveDog_Clicked(object sender, EventArgs e)
        {
            MessagingCenter.Send(this, "RemoveDog", Dog);
            await Navigation.PushModalAsync(new NavigationPage(new
CollectionPage()));
        }
```

Uvedený kód je pouze první částí. Tato část odešle do třídy s názvem „MockDataStorage“, která se stará o vykonání funkcí tlačítek, zprávu, že má být odstraněno určité plemeno.

```
MessagingCenter.Send(this, "RemoveDog", Dog);
```

V parametru si lze povšimnou, že je obsaženo „RemoveDog“, což je název metody, která má být vykonána. Po odstranění plemene se stránka naviguje na CollectionPage.

ZÁVĚR

Teoretická část začíná popisem nejčastěji používaných mobilních platforem. Dále se práce věnuje technologiím mobilního vývoje, jejich kladům a záporům. Práce se zaměřuje na multiplatformní vývoj pomocí frameworku Xamarin.Forms, především na jeho nové funkce, jako jsou Shell, CollectionView a CarouselView, u kterých je poukázáno i na staré způsoby a řešení.

V praktické části byla multiplatformním způsobem vytvořena mobilní aplikace, která zmíněné nové funkce demonstruje. V úvodu praktické části byl vytvořený nový Xamarin.Forms projekt s šablonou využívající Shell. Následně byly vytvořeny třídy pro práci s daty, které slouží pro vyplnění různých zobrazení v aplikaci daty. Další kapitola praktické části se věnuje a přibližuje využití Shell. Dále se práce věnuje CollectionView, způsobu, jakým lze upravit zobrazení položek, funkci SwipeView a její využití ve vytvořené aplikaci a použití RefreshView. Kapitola, která uzavírá praktickou část se nazývá CarouselView. Obsahuje popis, jakým způsobem CarouselView pracuje.

Hlavní přínos této práce spočívá v přiblížený funkcionality nových funkcí Xamarin.Forms a následné demonstraci jejich použití na reálné aplikaci.

SEZNAM POUŽITÉ LITERATURY

[1] LI, Wendong, Sumi HELAL a Raja BOSE. Mobile Platforms and Development Environments. February 2012. Morgan & Claypool Publishers, 2012. ISBN 978-1608458660.

[2] Mobile Operating System Market Share Worldwide [online]. [cit. 2020-03-30]. Dostupné z: https://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-200901-202003

[3] SHACKLES, Greg. Mobile development with C#. 1st ed. Sebastopol, CA: O'Reilly, c2012, xi, 155 p. ISBN 14-493-2023-6

[4] Mobilní platformy [online]. [cit. 2020-03-30]. Dostupné z: https://vyvojmobilnichaplikaci.com/mobilni-platformy/

[5] Android is Based on Linux, But What Does That Mean? [online]. [cit. 2020-03-30]. Dostupné z: https://www.howtogeek.com/189036/android-is-based-on-linux-but-what-does-that-mean/

[6] Distribution dashboard | Android [online]. [cit. 2020-03-30]. Dostupné z: https://developer.android.com/about/dashboards

[7] App Store [online]. [cit. 2020-03-31]. Dostupné z: https://developer.apple.com/support/app-store/

[8] CRAIGMILE, Natalie. Mobile App Platforms: Hybrid, Native, Mobile Web [online]. 2015 [cit. 2020-03-31]. Dostupné z: https://clutch.co/app-developers/resources/mobile-app-platforms-hybrid-native-mobile-web

[9] RICKARD, Chris. Choosing the right mobile app for your project: Native vs cross-platform vs hybrid [online]. 2016 [cit. 2020-03-31]. Dostupné z: http://inoutput.io/articles/development/choosing-the-right-mobile-app-for-your-project-native-vs-cross-platform-vs-hybrid

[10] HORBENKO, Yuliia. "Mobile Development: Choosing Between Native, Web, and Cross-Platform Applications [online]. [cit. 2020-03-31]. Dostupné z: https://steelkiwi.com/blog/how-choose-correct-platform-mobile-app-development/

[11] REHMANN, CHRISTOPH. Cross Platform Development mit Xamarin. Noser.com [online]. 2015, 03.03.2015 [cit. 2020-03-31]. Dostupné z: https://blog.noser.com/cross-platform-development-mit-xamarin/

[12] Univerzita Tomáše Bati ve Zlíně. Utb.cz [online]. [cit. 2020-03-31]. Dostupné z: https://www.utb.cz/

[13] Xamarin: for developers by developers. Devopedia [online]. 2018 [cit. 2020-07-28]. Dostupné z: https://devopedia.org/xamarin

[14] LIU, Shanhong. Cross-platform mobile frameworks used by software developers worldwide in 2019 and 2020 [online]. 2020, 2.6.2020 [cit. 2020-08-05]. Dostupné z: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/

[15] VIGNESH. Flutter vs. React Native vs. Xamarin: Detailed Comparison. Yourstory [online]. 2020, 19.3.2020 [cit. 2020-07-28]. Dostupné z: https://yourstory.com/mystory/flutter-vs-react-native-vs-xamarin

[16] KOROLEV, Sergey. Mobile App Development Approaches Explained. Railsware [online]. 2019, 19.6.2019 [cit. 2020-06-04]. Dostupné z: https://railsware.com/blog/native-vs-hybrid-vs-cross-platform/

[17] Dokumentace pro Xamarin. Microsoft [online]. [cit. 2020-06-04]. Dostupné z: https://docs.microsoft.com/cs-cz/xamarin/

[18] JHONSON, Justin, David BRITCH a Craig DUNN. What is Xamarin? Microsoft [online]. 2020, 28.5.2020 [cit. 2020-06-04]. Dostupné z: https://docs.microsoft.com/cs-cz/xamarin/get-started/what-is-xamarin

[19] Xamarin [online]. [2013] [cit. 2020-08-05]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Xamarin

[20] Xamarin: Úvod do Xamarin Forms. Skeleton: software [online]. 2017, 1.3.2017 [cit. 2020-08-05]. Dostupné z: https://www.skeleton.cz/uvod-do-xamarin-forms

[21] JHONSON, Justin. What is Xamarin.Forms. Microsoft [online]. 2020, 28.5.2020 [cit. 2020-08-05]. Dostupné z: https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin-forms

[22] ORTINAU, David. Xamarin.Forms 4.0: Getting Started with Shell. Microsoft [online]. 2019, 28.5.2019 [cit. 2020-08-05]. Dostupné z: https://devblogs.microsoft.com/xamarin/shell-xamarin-forms-4-0-getting-started/

[23] ORTINAU, David. Build .NET mobile apps quickly and easily with Xamarin.Forms Shell [online]. [cit. 2020-06-08]. Dostupné z: https://myignite.techcommunity.microsoft.com/sessions/85031

[24] JHONSON, Justin. Xamarin.Forms Shell Flyout. Microsoft [online]. 2020, 10.6.2020 [cit. 2020-06-08]. Dostupné z: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/shell/flyout?WT.mc_id=docs-xamarinblog-jamont

[25] JHONSON, Justin. Xamarin.Forms CollectionView. Microsoft [online]. 2019, 24.7.2019 [cit. 2020-06-01]. Dostupné z: https://docs.microsoft.com/cs-cz/xamarin/xamarin-forms/user-interface/collectionview/

[26] JHONSON, Justin. Xamarin.Forms CollectionView Data. Microsoft [online]. 2020, 29.04.2020 [cit. 2020-06-01]. Dostupné z: https://docs.microsoft.com/cs-cz/xamarin/xamarin-forms/user-interface/collectionview/populate-data

[27] JHONSON, Justin, David BRITCH a Craig DUNN. Customizing ListView Cell Appearance. Microsoft [online]. 2019, 9.12.2019 [cit. 2020-06-01]. Dostupné z: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/listview/customizing-cell-appearance

[28] JHONSON, Justin. Xamarin.Forms CarouselView Introduction. Microsoft [online]. 2019, 8.10.2019 [cit. 2020-06-01]. Dostupné z: https://docs.microsoft.com/cs-cz/xamarin/xamarin-forms/user-interface/carouselview/introduction

[29] JHONSON, Justin. Xamarin.Forms CarouselView Data. Microsoft [online]. 2020, 29.4.2020 [cit. 2020-06-15]. Dostupné z: https://docs.microsoft.com/cs-cz/xamarin/xamarin-forms/user-interface/carouselview/populate-data

[30] JHONSON, Justin. Xamarin.Forms Carousel Page. Microsoft [online]. 2017, 1.12.2017 [cit. 2020-06-15]. Dostupné z: https://docs.microsoft.com/cs-cz/xamarin/xamarin-forms/app-fundamentals/navigation/carousel-page

[31] Spokojený pes [online]. [cit. 2020-08-05]. Dostupné z: https://www.spokojenypes.cz/plemena-psu/

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

PWA Progresivní webová aplikace

C# Programovací jazyk platformy .Net

.Net Název platformy pro vývoj software

SDK Vývojové nástroje

IDE Vývojové prostředí

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PI CD – ROM

PŘÍLOHA PI

- Zdrojové kódy aplikace ve formátu .zip
- Práce ve formátu .pdf