Tvorba studijních materiálů ve frameworku Maui pro předmět Aplikační frameworky

Radomír Húdek

Bakalářská práce
2022

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Radomír Húdek**
Osobní číslo:	**A19038**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Tvorba studijních materiálů ve frameworku Maui pro předmět Aplikační frameworky**
Téma práce anglicky:	**Development of Study Materials in Maui Framework for the Course Application Frameworks**

Zásady pro vypracování

1. Popište současný stav technologií pro vývoj multiplatformních aplikací.
2. Zaměřte se především na framework Microsoft Maui.
3. Navrhněte zadání úkolů pro studenty zaměřené na framework Microsoft Mau.
4. Vytvořte navržené úkoly, popište jejich cíle, obsah, materiály a teorie k vypracování, vzorové řešení a možné kontrolní otázky.
5. Popište klíčové části navržených úkolů.
6. Demonstrujte výsledky a formulujte závěr.

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

Seznam doporučené literatury:

1. .NET documentation. Microsoft Docs [online]. Oficiálna dokumentácia firmy Microsoft Corporation [cit. 2021-10-18]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/
2. .NET Multi-platform App UI documentation. Microsoft Docs [online]. Oficiálna dokumentácia firmy Microsoft Corporation [cit. 2021-10-18]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/maui/
3. Xamarin documentation. Microsoft Docs [online]. Oficiálna dokumentácia firmy Microsoft Corporation [cit. 2021-10-18]. Dostupné z: https://docs.microsoft.com/en-us/xamarin/
4. PRICE, Mark J. C# 9 and .NET 5: Modern Cross-Platform Development: Build intelligent apps, websites, and services with Blazor, ASP.NET Core, and Entity Framework Core using Visual Studio Code. 5th Edition. Birmingham, UK: Packt Publishing, 2020. ISBN 978-1800568105.
5. PETZOLD, Charles. Creating Mobile Apps with Xamarin.Forms: Cross-platform C# programming for iOS, Android, and Windows. Redmond, Washington: Microsoft Press, 2016. ISBN 978-1-5093-0297-0.
6. HERMES, Dan. Xamarin Mobile Application Development: Cross-Platform C# and Xamarin.Forms Fundamentals. New York, New York, USA: Apress, 2015. ISBN 978-1484202159.

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

Datum zadání bakalářské práce: **3. prosince 2021**
Termín odevzdání bakalářské práce: **23. května 2022**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 24. ledna 2022

P**rohlašuji, 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;
- 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 – 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 17.05.2022

Radomír Húdek v.r.

podpis studenta

ABSTRAKT

Bakalárska práca sa zaoberá technológiami multiplatformového vývoja a frameworkom .NET MAUI. V práci sú diskutované mobilné zariadenia a najpoužívanejšie operačné systémy. Ďalej sú popísane druhy mobilných aplikácii, a to natívne a multiplatformové. Náplňou praktickej časti práce je tvorba krátkych úloh pre precvičenie jednotlivých častí tvorby aplikácie a príprava prostredia pre prácu vo frameworku .NET MAUI.

Kľúčové slová: .NET MAUI, .NET, C#, XAML, multiplatformové aplikácie, multiplatformové nástroje, úlohy, Visual Studio, mobilné zariadenie

ABSTRACT

The Bachelor's thesis is focused on multiplatform development technologies and .NET MAUI framewok. The work discusses mobile devices and the most commonly used operating systems. The types of mobile applications, namely native and multiplatform, are described. In the practical part of the work is creation of short tasks for practicing the individual parts of application development and preparing the environment for working in the .NET MAUI framework.

Keywords: .NET MAUI, .NET, C#, XAML, multiplatform apps, cross-platform tools, exercises, Visual Studio, mobile device

Chcel by som sa poďakovať vedúcemu práce Ing. et Ing. Erikovi Královi, Ph.D. za odborné vedenie, cenné rady a konzultácie bakalárskej práce.

OBSAH

ÚVOD

Cieľom práce je popísať súčasný stav vývoja multiplatformových mobilných aplikácií. Multiplatformový vývoj sa stal veľmi dôležitou súčasťou moderného vývoja aplikácii, pretože dokáže ušetriť nie len čas potrebný na vývoj, ale taktiež finančné zdroje, pretože ak by sa aplikácia vyvíjala natívne, bola by potreba mať tím vývojárov pre každú platformu zvlášť.

Práve preto vznikol aj framework, od spoločnosti Microsoft, založený na technológii .NET, a taktiež je označovaný ako nepriamy následník frameworku Xamarin.Forms. .NET MAUI, ako nesie názov, bude hlavným bodom tejto bakalárskej práce, kde sa čitateľ oboznámi s architektúrou, ako funguje tento framework a ktoré operačné systémy a ich verzie sú podporované.

Významnú rolu vo vývoji multiplatformových aplikácii hrajú mobilné zariadenia a operačné systémy. Momentálne existujú dva najpoužívanejšie mobilné operačné systémy, pre ktoré sú aplikácie najmä vyvíjané, Android a iOS. Okrem natívneho vývoja existuje aj vývoj hybridných a webových aplikácii, ktoré spadajú pod multiplatformový vývoj.

V praktickej časti tejto práce sú demonštrované krátke úlohy na tvorbu jednotlivých častí aplikácie pomocou frameworku .NET MAUI.

I. TEORETICKÁ ČÁST

1 MOBILNÉ ZARIADENIA

Mobilné zariadenie je vreckový tablet alebo iné zariadenie, ktoré je vyrobené pre prenositeľnosť a preto je kompaktné a ľahké. Nové technológie pre ukladanie, spracovanie a zobrazovanie dát umožnili týmto malým zariadeniam robiť takmer všetko, čo sa už skôr dalo robiť s väčšími stolovými počítačmi. [20]

Práve väčšina osobných počítačov sa v dnešnej dobe vyskytuje vo forme menšieho zariadenia. V tejto forme sa najmä využívajú na rýchle získanie informácii, média a sociálne siete. Tablety a smartfóny majú zásadne rozdielnu používateľskú interakciu so zariadením oproti desktopovým zariadeniam, ktorá je založená primárne na dotyku, virtuálna klávesnica sa zobrazí len v nevyhnutných prípadoch. [1]

Jedna z kľúčových vlastností mobilného zariadenia je schopnosť byť informovaný o aktuálnom prostredí pomocou vstavaných senzorov. Tieto zariadenia majú senzory navrhnuté tak, aby zachytili pomerne všetky možné informácie z prostredia okolo nich. [2]

Ďalšia veľmi dôležitá vlastnosť mobilného zariadenia je schopnosť komunikácie s ostatnými výpočtovými zariadeniami prostredníctvom rôznych mechanizmov, ako sú Wi-Fi, Bluetooth, mobilná sieť a NFC. [2]

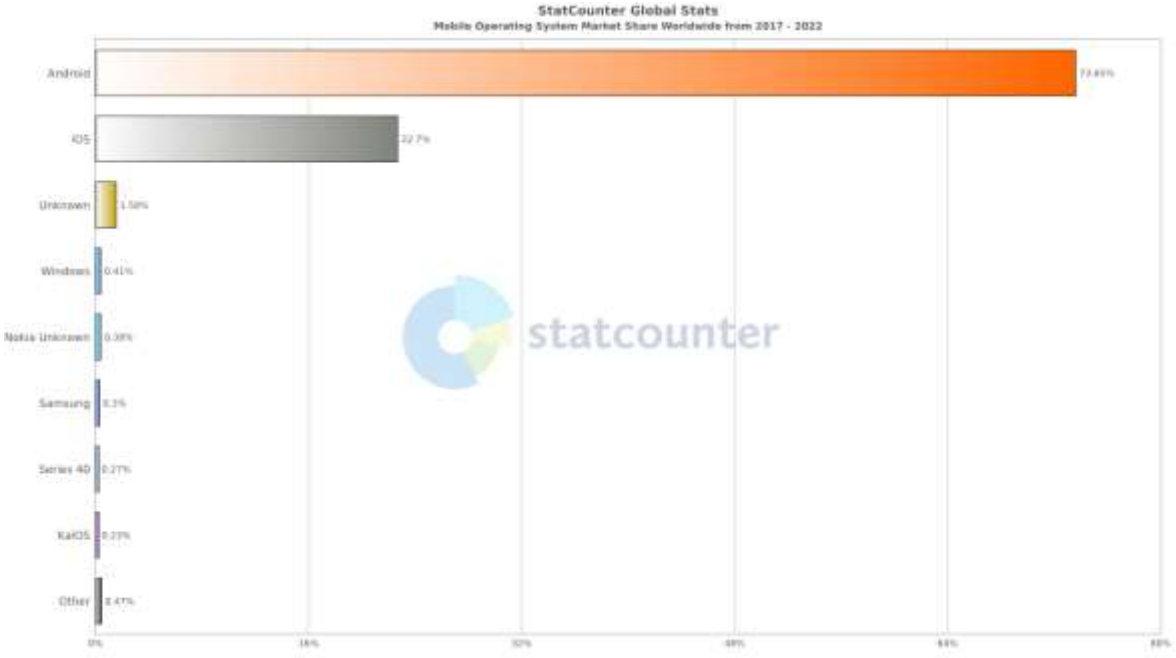
Okrem funkcií, ktoré nie sú dostupné v iných počítačových platformách, mobilné zariadenia majú väčšinu funkcií rovnakých. Niektoré funkcie majú lepšiu použiteľnosť, ako napríklad kamera. Zadávanie údajov alebo textu je tiež podobné, používateľ môže použiť klávesnicu, hlasové zadávanie alebo dať systému pokyn aby niečo zadal sám. Mobilné zariadenia dokážu taktiež ukladať údaje v mnohých formátoch vrátane relačných databáz. [2]

1.1 Operačný systém

Operačný systém je softvér, ktorý slúži ako rozhranie medzi hardvérovými komponentami počítača a konečným používateľom. Každý počítač musí mať aspoň jeden operačný systém pre beh ostatných programov. Aplikácie potrebujú prostredie kde budú bežať a vykonávať svoje úlohy. Nie je možné pre používateľa používať počítač alebo mobilné zariadenie bez toho aby obsahovali operačný systém. [21]

Napriek tomu, že trh s mobilnými zariadeniami má potenciál sa rýchlo meniť, prevládajú momentálne dva operačné systémy pre ktoré sa vyvíjajú aplikácie, a to iOS a Android. [1]

Podľa internetovej stránky StatCounter, Obrázek 1 Podieľ operačných systémov na celosvetovom trhu [3], za posledných 5 rokov vyzeral podiel operačných systémov celosvetovo v prepočte asi 73,65 % Android užívateľov a 22,7 % používateľov zariadení iOS. Ostatné



Obrázek 1 Podieľ operačných systémov na celosvetovom trhu [3]

operačné systémy tvorili okolo 3,5 % užívateľov.

1.2 Operačný systém Android

Je populárny mobilný operačný systém postavený na jadre Linux, vytvorený firmou Google.[4] Primárne ho nájdeme ako základný operačný systém mobilných zariadení a tabletov. Je to open-source OS, ktorý ho robí zadarmo prístupným každému, dokonca aj pre komerčné účely. Toto ho robí veľmi výnimočným oproti ostatným OS, ktoré nie sú zadarmo prístupné.[5]

1.2.1 Architektúra Android

Obrázek 2 Architektúra operačného systému Android [22]

Architektúru Androidu, ktorú môžeme vidieť na obrázku 2, je zložená z komponentov na podporu potrieb mobilných zariadení. Tieto komponenty môžeme rozložiť na päť vrstiev, Systémové aplikácie, Android framework, Android Runtime a natívne C/C++ knižnice, Hardware Abstraction Layer a posledná vrstva jadro Linux.

Linux kernel je základom Android platformy. Android Runtime je závislé na linuxovom jadre kvôli základným funkciám ako sú vlákna a nízko-úrovňové riadenia pamäti. Používanie linux jadra povoľuje Androidu využiť kľúčové výhody zabezpečovacích funkcií a povoľuje výrobcom zariadení vytvoriť hardvérové ovládače pre toto jadro. [6]

Abstraktná vrstva hardvéru (HAL) poskytuje štandardne rozhranie, ktoré vystavujú hardvér pre Java API frameworky. HAL sa skladá z viacerých modulov knižníc, z ktorých každá obsahuje rozhranie pre typický hardvérový komponent. Keď API potrebuje prístup k hardvérovému zariadeniu, systém načíta knižnicu pre daný modul. [6]

Pre zariadenie, ktoré beží na verzií Android 5.0 alebo vyššie, každá aplikácia spustí vlastný proces s vlastnou instanciou ART. ART je napísaný tak, aby mohol spustiť viac virtuálnych zariadení na nízko-pamäťovom zariadení spustením DEX súborov vo formáte bytecode, ktorý je navrhnutý špeciálne pre Android a optimalizovaný pre minimálnu pamäťovú stopu. [6]

Najhlavnejšie funkcie ART [6]:

- Kompilovanie dopredu (AOT) a kompilovanie za behu (JIT)
- Lepšia podpora ladenia

Mnoho základných komponentov ako ART a HAL, sú vytvorené z natívneho kódu, ktorý vyžaduje natívne knižnice napísané v jazyku C a C++. Java framework API vystavuje funkcionalitu týchto natívnych knižníc aplikáciám. [6]

Celá sada funkcií OS Androidu je k dispozícii prostredníctvom API napísanej v jazyku Java. Táto API je základom pre tvorbu android aplikácii. [6]

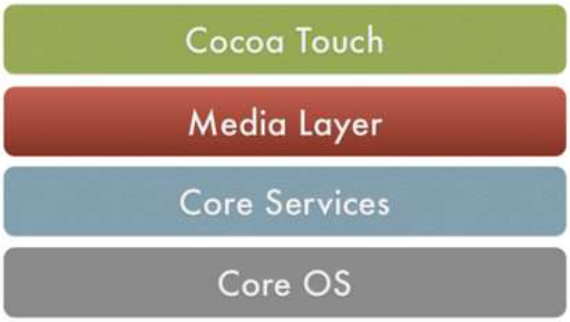
Android prichádza so sadou základných systémových aplikácii. Tieto aplikácie nemajú žiadny dôležitý status, takže používateľ sa môže rozhodnúť nahradiť tieto aplikácie aplikáciami tretích strán. [6]

1.3 Operačný systém iOS

Je mobilný operačný systém pre zariadenia vyrobené spoločnosťou Apple. iOS beží na zariadeniach ako iPhone, iPad, iPod Touch a Apple TV. [23]

iOS je známy tým, že slúži ako základný softvér, ktorý umožňuje používateľom iPhone komunikovať so svojim telefónom pomocou gest, ako je potiahnutie prstom, ťuknutie a zovretie. Tieto akcie vykonané pomocou prstov sa väčšinou vykonávajú na viacdotykových displejoch, ktoré poskytujú rýchlu odozvu a prijímajú vstupy z viacerých prstov. Hoci to nie je mobilný operačný systém číslo jedna na celom svete, iOS dominuje severoamerickým trhom. [23]

1.3.1 Architektúra iOS



Obrázek 3 Architektúra operačného systému iOS [24]

iOS je odvodený od Mac OS X a je založený na Unix operačnom systéme. Skladá sa zo štyroch abstrakčných vrstiev, ktoré môžeme vidieť na obrázku 3:

- Core OS: Poskytuje nízko-úrovňové funkcie ako sú frameworky pre bezpečnosť a interakciu s externým hardvérom.
- Core Services: Poskytuje služby požadované vrchnými vrstvami.
- Media Layer: Poskytuje potrebné technológie pre grafiku, video a audio.
- Coca Touch: Nachádzajú sa tu frameworky, ktoré sú často používané pri tvorbe aplikácii. [23]

iOS prichádza s predinštalovanými aplikáciami ako je email klient, webový prehliadač Safari, kontakty a prehrávač médií. [23]

Vývojári môžu využívať iOS SDK pre tvorbu aplikácií pre mobilné zariadenia Apple. SDK obsahuje nástroje a rozhrania na vývoj, inštaláciu, spustenie a testovanie aplikácii. Natívne aplikácie môžu byť napísane pomocou iOS frameworkov a Objective C programovacieho jazyku. Súčasťou iOS SDK sú Xcode Tools, ktoré zahŕňajú integrované vývojové prostredie pre správu aplikačných projektov a grafický nástroj pre tvorbu UI a ladiaci nástroj na analýzu výkonnosti. Taktiež obsahuje iOS simulátor, ktorý povoľuje vývojárom testovanie aplikácie na Mac a vývojársku knižnicu iOS, ktorá poskytuje všetky potrebné dokumenty a referenčné materiály. [23]

2 VÝVOJ MOBILNÝCH APLIKÁCII

Vývoj mobilných aplikácii je proces tvorby softvéru pre smartfóny a digitálne zariadenia, najčastejšie pre Android a iOS.[7] Mobilné aplikácie slúžia na poskytovanie podobných funkcií používateľom ako na desktop zariadení. [25]

2.1 Natívne mobilné aplikácie

Natívny vývoj aplikácie znamená, že aplikácia je vytvorená pre špecifickú platformu. Tento prístup vývoja aplikácií sa používa hlavne pre veľké projekty, kde časová náročnosť a finančné náklady nehrajú žiadnu rolu pri vývoji a je potrebné dosiahnuť čo najlepší možný výkon.

Natívne aplikácie operačného systému Android sú vyvíjané v jazyku Java alebo Kotlin, v prípade iOS aplikácii sa jedná o Objective C a Swift a pre Windows C# a XAML. Natívny vývoj je ideálny, keď chceme používateľovi dopriať najlepšie možné riešenie čo sa týka UI a dizajnu aplikácie. Vďaka tomuto vývoju dokážu vývojári zlepšiť beh a funkcie aplikácii, pretože majú priamy prístup k operačnému systému.[8]

2.2 Multiplatformové mobilné aplikácie

Mobilné aplikácie sa stávajú nevyhnutné pre potreby rôznych odvetví priemyslu a širokú škálu firiem. Ich najväčšou výhodou je teda to, že aplikácie dokážu fungovať na viacerých mobilných platformách súčasne ako iOS a Android.[9]

Vďaka tejto technológii sme zvýhodnený v tom, že nepotrebujeme mať separátny tím vývojárov pre každú platformu zvlášť, pretože máme umožnené spustiť a ďalej vyvíjať softvér pomocou veľkého množstva multiplatformových vývojárskych nástrojov. Tento vývoj sa teda stal dôležitou alternatívou tradičnému natívnemu vývoju.[10]

Aplikácie vyvíjané touto technológiou majú natívny dizajn platformy na ktorej je spustený, ale aj funkčnosť, ktorá je v dôsledku spojenia natívneho kódu a nezávislého, ktorý je kompatibilný pre viaceré platformy.[9]

2.2.1 Webové aplikácie

Webová aplikácia je uložená na serveri a komunikuje prostredníctvom internetu cez prehliadačove rozhranie. Aby webová aplikácia fungovala potrebujeme webový server, aplikačný server a databázu. Webový server spravuje požiadavky, ktoré prichádzajú od klienta, aplikačný server dokončí požadovanú úlohu. K uloženiu akýchkoľvek potrebných informácii je prítomná databáza.[38]

Výhody

- Finančne a časovo šetrný vývoj v porovnaní s ostatnými druhmi aplikácii.
- Podpora akéhokoľvek operačného systému pod podmienkou webového prehliadača.
- Nie je potreba častej aktualizácie, nakoľko je aplikácia priamo napojená na web a má vždy aktuálnu verziu.
- Jednoduchšia úprava webovej aplikácie z dôvodu jednoduchšej možnosti zmeny rozhrania.
- Nie je potrebné aplikáciu sťahovať, spustenie prebieha vo webovom prehliadači.[39]

Nevýhody

- Priama väzba na webový prehliadač, toto sa môže odzrkadliť na výkone aplikácii.
- Závislosť na internetovom pripojení a webe, ak web zlyhá, zlyhá aj aplikácia.
- Obťažnejšie získať povedomie publika. [39]

2.2.2 Hybridné aplikácie

Hybridná aplikácia je taká, ktorá využíva natívne aj webové funkcie na poskytovanie jednotného UX v rámci jednej aplikácie. Hybridné aplikácie môžu ponúknuť to najlepšie z architektúr oboch prístupov, no nesú so sebou aj množstvo nevýhod. [11]

Aplikácie, ktoré sú vytvorené hybridným prístupom sú napísané v jazykoch ako CSS, HTML a JavaScript. Tieto aplikácie potrebujú vývojári spraviť ako natívnu aplikáciu, to dosiahnu pomocou Ionic Capacitor alebo Apache Cordova. Vďaka tomu môže byť zdrojový kód napísaný len raz a ten istý kód môže byť následne prispôsobený pre rôzne platformy. [8]

Výhody

- Taktiež finančne a časovo šetrný vďaka jednotnej kódovej základne.
- Nie je tu verzovanie aplikácie, čiže nie je potrebné vytvárať novú aplikáciu keď vyjde nový operačný systém.
- Ľahká škálovateľnosť na rôzne platformy.
- Pracujú v offline režime.
- Neustále aktualizácie, ktoré opravujú chyby.[40]

Nevýhody

- Slabší výkon, pretože medzi operačným systémom a zdrojovým kódom operuje jedna vrstva.
- Niektoré natívne funkcie nie sú k dispozícii.
- Používateľské rozhranie je z pohľadu UX zlé.
- Tendencia k chybám.
- Potrebné spustiť aplikáciu pomocou webového prehliadača. [40]

3 TECHNOLÓGIE MULTIPLATFORMOVÉHO VÝVOJA

Multiplatformový vývoj môžeme pochopiť ako vývoj softveru, ktorého výsledný produkt môžeme spustiť na viac ako jednej platforme alebo operačnom systéme. V súčasnej dobe existuje množstvo nástrojov a technológií, ktoré uľahčujú vývoj aplikácii.

3.1 .NET

Je bezplatná, multiplatformová, open source vývojárska platforma pre vývoj rôznych druhov aplikácii. Pomocou .NET môžeme využiť množstvo jazykov, editorov a knižníc pre tvorbu webových, mobilných, desktopových aplikácii. [27]

.NET aplikácie sa dajú písať v jazykoch C#, F# a Visual Basic.

- C# je jednoduchý, moderný, objektovo orientovaný programovací jazyk
- F# je programovací jazyk pre tvorbu robustných kódov
- Visual Basic je jazyk s jednoduchou syntaxou pre tvorbu objektovo orientovaných aplikácii [27]

3.1.1 Implementácie .NET

Každá implementácia dovoľuje .NET kódu pracovať na iných platformách. Doteraz sú známe 3 implementácie. [28]

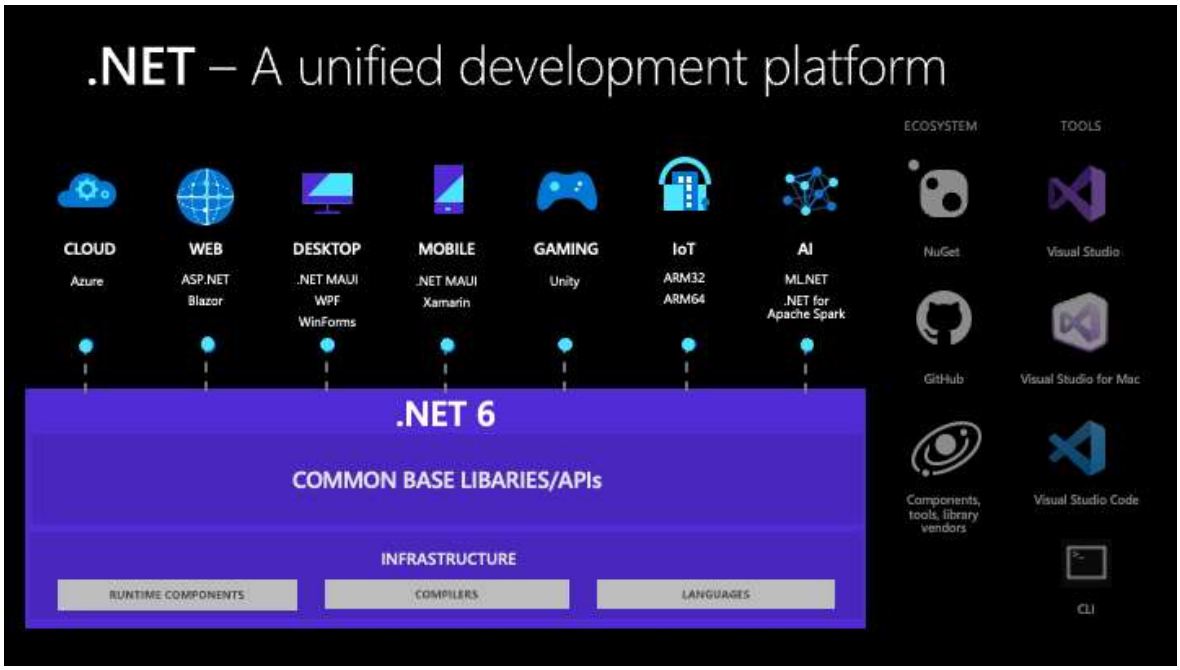
.NET Framework je pôvodná implementácia .NET. Podporuje beh webstránok, služieb, desktopových aplikácii na Windows. [28]

.NET je multiplatformová implementácia pre beh webstránok, služieb a konzolových aplikácii pre Windows, Linux a macOS. Predchádzajúci názov bol .NET Core. [28]

Xamarin/Mono je implementácia pre beh aplikácii na mobilných operačných systémoch. [28]

3.1.2 .NET 6

.NET 6 prináša zjednotenie implementácii, ktoré sa začali v .NET 5. Taktiež prináša zjednotenie SDK, knižníc a prostredia pre beh naprieč všetkými druhmi aplikácii. [19]



Obrázek 4 Grafické zobrazenie zjednotenej platformy .NET 6 [26]

3.2 Xamarin.Forms

Xamarin je open-source platforma pre vytváranie moderných a výkonných aplikácii pre Android, iOS a Windows s využitím .NET.[29]

Xamarin.Forms je open-source UI framework, ktorý umožňuje vývojárom vytvárať aplikácie Xamarin.Android, Xamarin.iOS a Windows z jednej zdieľanej kódovej základne. [29]

Umožňuje vývojárom vytvárať UI v jazyku XAML a s kódom v pozadí v jazyku C#. Tieto rozhrania sú vykreslené ako výkonné natívne ovládacie prvky na každej platforme. [29]

Tento framework je pre vývojárov, ktorý hľadajú nasledovné:

- Zdieľaný dizajn a rozvrhnutie UI naprieč platformami.
- Zdieľaný kód a testovanie naprieč platformami.
- Písanie aplikácii pre rôzne platformy v jazyku C# za pomoci Visual Studia. [29]

Podrobnejšie informácie ohľadom Xamarin a Xamarin.Forms nájdeme v oficiálnej dokumentácii. [29]

3.3 Flutter

Flutter je open-source mobilný UI framework vytvorený firmou Google. Umožňuje vytvoriť natívnu mobilnú aplikáciu iba s jednou kódovou základňou. To znamená, že môžeme použiť jeden programovací jazyk a jednu kódovú základňu k vytvoreniu dvoch rôznych aplikácii, pre iOS a Android. [30]

Flutter sa skladá z dvoch dôležitých častí:

- SDK: Súbor nástrojov, ktoré pomáhajú vyvíjať aplikácie. To zahŕňa nástroje pre kompiláciu kódu do natívneho strojového kódu.
- Framework (Knižnica UI založená na widgetoch): Zbierka opakovane používateľných prvkov UI, ako sú tlačítka, textové vstupy, atď., ktoré môžeme prispôsobiť pre svoje vlastné potreby. [30]

Pre vývoj sa používa programovací jazyk Dart. Dart sa zameriava na front-end vývoj a môže byť použitý k tvorbe mobilných a webových aplikácii. Dart je typovo objektový programovací jazyk. Syntaxia Dartu sa môže porovnať so syntaxiou JavaScriptu. [30]

Detailnejšie informácie ohľadom tohto frameworku sa vyskytujú na oficiálnej dokumentácii Flutter. [32][31]

3.4 ASP .NET Core Blazor

Blazor je framework, ktorý nám dovoľuje vyvíjať interaktívne webové UI používaním jazyka C# miesto JavaScript. Zdieľanie aplikačnej logiky na strane serveru a klienta prebieha pomocou .NET. Vykresľovanie UI prebieha v jazyku HMTL a CSS pre širokú podporu prehliadačov. Tvorbu hybridných desktopových a mobilných aplikácii.[33]

Blazor používa webové štandardy bez pluginov a transpilácie[1] kódu. Kód bežiaci v prehliadači je spustený v rovnakom rámci bezpečnosti ako JS frameworky. Blazor kód spustený na serveri je dostatočne flexibilný na to, aby sa správal „normálne“ na serveri, čiže napríklad priamy prístup do databáze.[34]

[1] Transpilácia je proces konvertovania kódu z jedného jazyka do druhého.

3.5 Ionic

Ionic je open-source sada nástrojov UI pre vytváranie výkonných, vysoko kvalitných mobilných a desktopových aplikácii pomocou webových technológii ako sú HTML, CSS a JavaScript, a s integráciou pre obľúbené frameworky ako Angular, React a Vue.[35]

Ionic sa zameriava na UX a UI aplikácie ako sú ovládanie UI, interakcie, gesta a animácie. Alternatívne sa dá použiť aj samostatne bez front frameworku pomocou jednoduchého skriptu. [35]

Ionic je jediná sada mobilných aplikácii, ktorá umožňuje webovým vývojárom vytvárať aplikácie a mobilný web z jedinej kódovej základne. Vďaka Adaptive Styling vypadajú aplikácie ako keby boli natívne. [35]

Podrobnejšie informácie sa vyskytujú na oficiálnej dokumentácii Ionic. [35]

3.5.1 Ionic Capacitor

Je to multiplatformný natívny runtime, ktorý uľahčuje vytváranie moderných webových aplikácii, ktoré bežia natívne na iOS, Android a webe. Predstavuje evolúciu hybridných aplikácii a vytvára webové natívne aplikácie, ktoré poskytujú moderný natívny kontajnerový prístup pre tímy, ktoré chcú vytvoriť ako prvé web bez toho aby obetovali plný prístup k natívnemu SDK. [36]

Poskytuje konzistentnú sadu rozhraní API zameranú na web, ktorá aplikáciám umožní zostať čo najbližšie webovým štandardom a zároveň mať prístup k bohatým natívnym funkciám zariadenia na platformách, ktoré podporujú. Pridanie natívnych funkcií je jednoduché vďaka Plugin API pre Swift na iOS, Java na Androide a JavaScriptu pre web. [36]

Pre podrobnejšie informácie a fungovanie treba navštíviť oficiálnu dokumentáciu.[36]

3.6 React Native

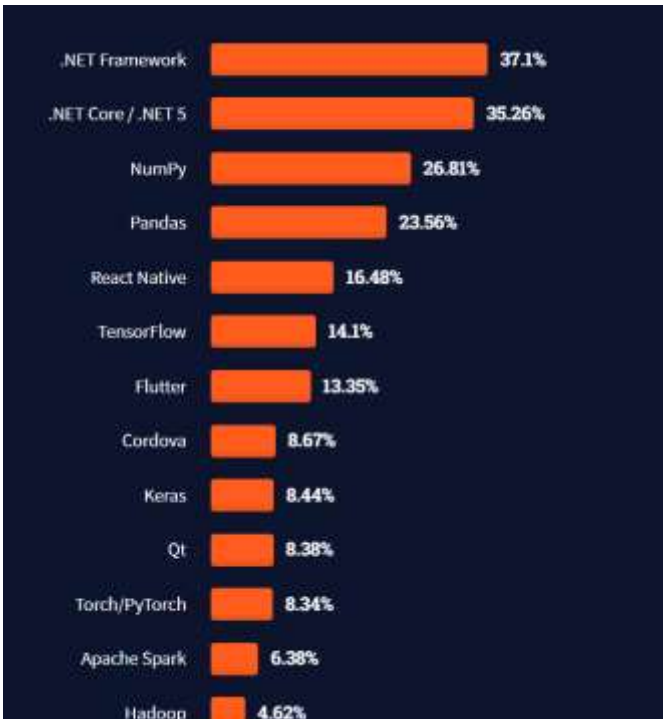
Je open-source framework, pre vývoj mobilných aplikácii, ktorý umožňuje vývoj multiplatformových Android, iOS a webových aplikácii. Tento framework je založený na knižnici React od Facebook-u, čiže je založený na programovacom jazyku JavaScript.[41]

Multiplatformové aplikácie sú umožnené vytvárať vďaka jednotnej kódovej základne. React Native následne kompiluje tento kód do natívnych komponentov použitím API a modulov špecifických pre platformu. Architektúra využíva konceptu „mostu“, ktorý povoľuje asynchrónnu komunikáciu medzi JS a natívnymi prvkami. Tento druh architektúry povoľuje používanie veľkého množstva natívnych prvkov, ale na druhej strane konštantné užívanie „mostu“ značne znižuje výkon aplikácie.[42]

Pre viac informácii ohľadom tohto frameworku navštívte oficiálnu dokumentáciu.[43]

3.7 Najpoužívanejšie technológie multiplatformového vývoja

Podľa prieskumu stránky Stack Overflow (Obr. 5), patrí .NET Framework a .NET Core/.NET 5 medzi najpoužívanejšie frameworky medzi profesionálnymi vývojármi. Obľúbenosti sa tešili aj ďalšie spomenuté frameworky ako React Native a Flutter.



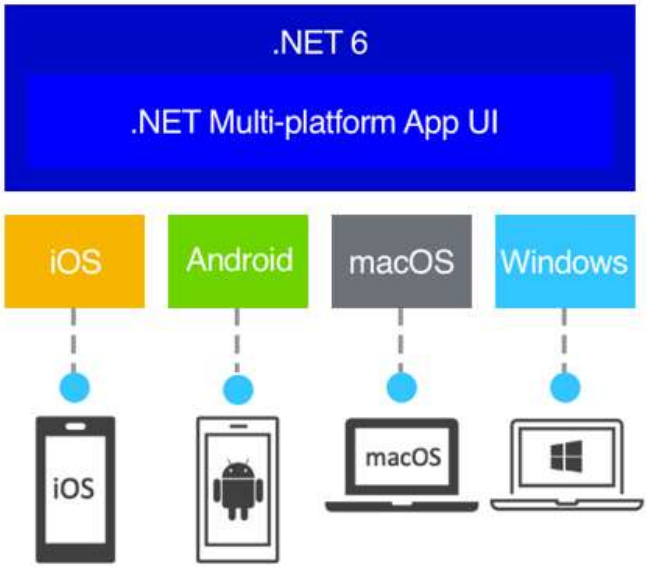
Obrázek 5 Graf najpoužívanejších frameworkov a knihovien medzi profesionálnymi vývojármi[37]

4 .NET MAUI

.NET Multi-platform APP UI alebo teda .NET MAUI, nám dáva možnosť vyvíjať aplikácie pomocou .NET multiplatformového UI nástroja, ktorý sa zameriava na platformy mobilných a desktopových zariadení ako sú Android, iOS, macOS a Windows. Vývoj prebieha v jazyku C# a XAML. [12]

Pre správnu fungovanie a vyvíjanie v tomto frameworku je potreba mať k dispozícii najnovší .NET 6. [13]

Z obrázku 6 môžeme vyčítať, že tento framework od spoločnosti Microsoft nám poskytuje možnosť vyvíjať aplikácie, ktoré dokážu fungovať zároveň na viacerých platformách vďaka jednotnému zdieľanému základnému kódu. [12]



Obrázek 6 Platformy frameworku .NET MAUI [8]

.NET MAUI je open-source framework a je nástupcom Xamarin.Forms, ktorý je rozšírený o platformy stolových zariadení. Vďaka UI kontrolérom je prerobený od základov pre lepší výkon a škálovateľnosť na platformách. Ak sa niekto stretol a používal Xamarin.Forms, tak si všimne isté podobnosti týchto frameworkov. Samozrejme sú tu aj rozdiely. Pomocou .NET MAUI môžeme vytvárať multiplatformové aplikácie používaním jedného projektu, ale môžeme pridať zdrojový kód a prostriedky špecifické pre danú platformu, ak je to potrebné. Hlavný zámer tohto frameworku je, že nám dáva možnosť implementovať najviac ako je len možné aplikačnej logiky a rozloženie používateľského rozhranie v jednotnom kóde. [12]

4.1.1 Ako pracuje tento framework

.NET MAUI zjednocuje API Androidu, iOS, macOS a Windows do jednej API, ktorá umožňuje vývojárovi napísať kód jeden-krát, zatiaľ čo umožňuje prístup ku každému aspektu natívnej platformy. [14]

.NET 6 poskytuje sériu frameworkov pre vytváranie aplikácii určených pre každú platformu zvlášť a to .NET pre Android, .NET pre iOS, .NET pre macOS a Windows UI 3. [14]

Tieto frameworky majú prístup k rovnakej knihovne rozhrania .NET 6, Basic Class Library. Táto knihovňa obsahuje detaily platformy rozdielne od nášho kódu. BCL je závislá na module .NET Runtime, ktorý poskytuje spúšťacie prostredie pre náš kód. Android, iOS a macOS majú prostredie implementované pomocou Mono, implementáciou .NET Runtime. Pre Windows spúšťacie prostredie poskytuje Win32. [14]

Každá platforma obsahuje iné možnosti definovania používateľského rozhrania pre aplikáciu a poskytuje rôzne variácie modelov, ako budú prvky používateľského prostredia komunikovať a vzájomne spolupracovať. Používateľské prostredie môže byť vytvorené pre každú platformu zvlášť použitím frameworkov vhodných pre danú platformu, ale toto potrebuje aby sme zachovali natívny kód pre každý druh zariadení. [14]

4.1.2 Architektúra .NET MAUI

Na nasledujúcom obrázku, , je poukázaný detailný pohľad na architektúru .NET MAUI aplikácie.

Obrázek 7 Architektúra .NET MAUI aplikácie [12]

V .NET MAUI napíšeme kód, ktorý primárne komunikuje s API .NET MAUI, bod 1 na obrázku 7. .NET MAUI potom priamo pracuje s API natívnej platformy, bod 3 na obrázku 7. Ak je potreba, tak kód môže byť priamo spustený na natívnej API, bod 2 na obrázku 7. [14]

Aplikácie môžu byť vyvíjané na PC alebo Mac a byť kompilované do natívnych aplikačných balíčkov:

- Android aplikácie sa kompilujú z programovacieho jazyka C# do prechodného jazyka (IL), ktoré sú potom kompilované za behu (JIT) do natívnej podoby aplikácie.
- iOS aplikácie sú plne dopredu kompilované (AOT) z C# do natívneho kódu zostavenia ARM.
- macOS aplikácie k vývoju používajú riešenie od Apple, mac Catalyst, ktoré prináša iOS aplikácie vytvorené pomocou UI Kit na desktop a sú rozšírené o prídavne rozhranie AppKit a API platformy, ak je potreba.
- Windows aplikácie sú vytvorené pomocou Windows UI 3. [14]

4.1.3 Podporované platformy

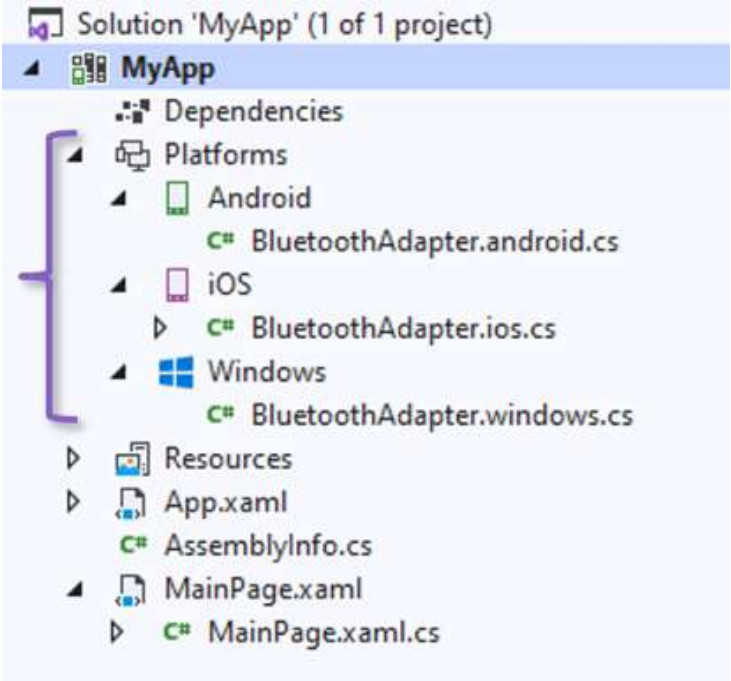
Aplikácie .NET MAUI sú podporované pre tieto platformy:

- Android 5.0 (API 21) alebo vyšší.
- iOS 10 alebo vyšší.
- macOS 10.13 alebo vyšší
- Windows

Mimo tieto platformy majú podporu aj operačné systémy Tizen od spoločnosti Samsung a Linux. [15]

4.1.4 .NET MAUI projekt

Jeden projekt preberá špecifické vývojové prostredie pre platformu a „zabaľuje“ ich do jedného zdieľaného projektu, ktorý môže cieliť na Android, iOS, macOS a Windows zároveň. [17]



Obrázek 8 Detailný prehľad súborov v projekte [17]

Poskytnutie zjednodušeného a konzistentného prostredia pre vývoj nám poskytuje nasledujúce funkcie:

- Jeden zdieľaný projekt.
- Zjednodušené ladenia.
- Zdieľané súbory prostriedkov.
- Prístup k API a nástrojom špecifických pre konkrétnu platformu.
- Jeden vstupný bod aplikácie. [17]

II. PRAKTICKÁ ČÁST

5 PRÍPRAVA PROSTREDIA

Pre tvorbu aplikácie .NET Multi-platform App UI, skrátene .NET MAUI sa požaduje najnovšia preview verzia Visual Studia 2022 a .NET 6.

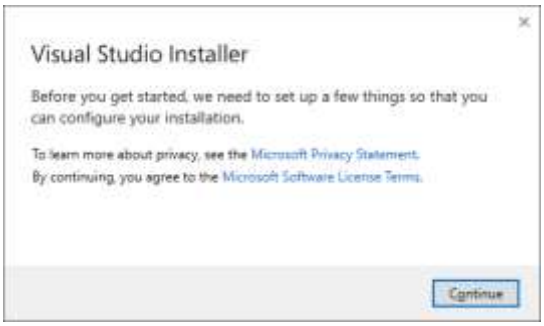
5.1 Visual Studio

Pre prvé spustenie a vytvorenie aplikácie je potrebné stiahnuť a nainštalovať najnovšiu verzia Visual Studio 2022 Preview. Inštaláciu Visual Studia spustiť po stiahnutí inštalačného súboru, ktorý by mal niesť jeden z daných názvov:

- vs_community – verzia, ktorá je vhodná pre potreby študenta
- vs_professional
- vs_enterprise

Ak sa zobrazí upozornenie o kontrole používateľských kont potvrdiť ÁNO/YES.

Pred inštaláciou sa zobrazí dialóg o prečítaní licenčných podmienok a vyhlásenie spoločnosti Microsoft o ochrane osobných údajov (Obr. 9).



Obrázek 9 Dialógové okno s licenčnými podmienkami a s vyhlásením o ochrane osobných údajov

Po potvrdení dialógového okna sa nainštaluje VS Installer, kde sa zobrazí prvá zo štyroch možných záložiek:

- Workloads
- Individual components
- Language packs
- Installation locations

V záložke Workloads sa zvolia potrebné workloady, ktoré budú zaručovať správnu funkčnosť frameworku .NET MAUI. Workloady (Obr. 10), ktoré sú potrebné sa nachádzajú v časti Desktop & Mobile a sú to:

- Mobile development with .NET
- Universal Windows Platform development

Obrázek 10 Workloads, ktoré su nevyhnutné pre správnu funkčnosť .NET MAUI [13]

Po zvolení potrebných workloads sa ešte treba uistiť, že je zvolený aj požadovaný .NET MAUI (Prewiev) (Obr. 11), bez ktorého nebude možné vytvoriť projekt.

Obrázek 11 Detaily inštalácie

V Individual components je na výber množstvo ďalších komponentov pre prácu, ale tie komponenty, ktoré sú požadované už sú automaticky predvolené.

V Language packs sa vyberie jazyk, ktorým bude prostredie disponovať, odporúčaný jazyk je ANGLIČTINA – English.

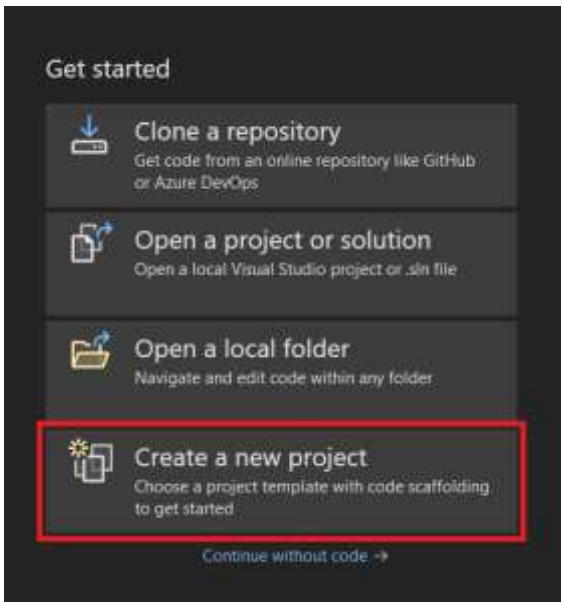
Záložka Installation locations slúži pre výber umiestnenia, kde sa VS nainštaluje.

Po úspešnej inštalácii Visual Studio 2022 Preview by malo byť umožnené spustiť VS.

5.2 Prvé vytvorenie aplikácie .NET MAUI

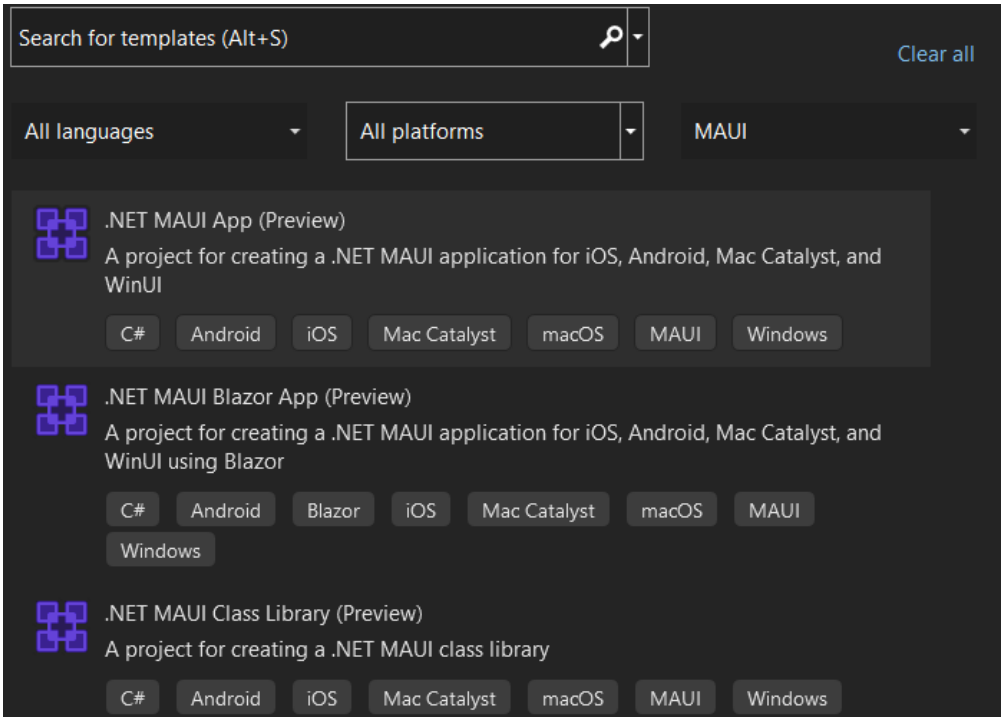
Po úspešnej inštalácii a spustený VS je potreba vytvoriť prvý projekt. Po spustený VS sa zobrazí dialógové okno, kde bude na výber z viacero možností a to:

- Clone a repository
- Open a project or solution
- Open a local folder
- Create a new project



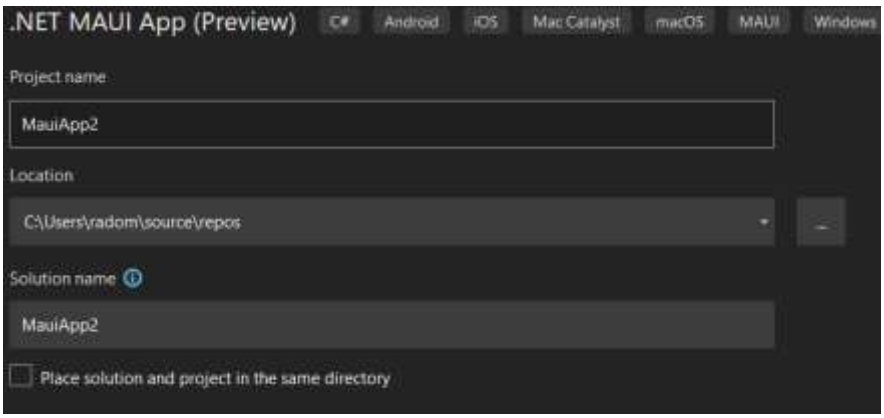
Obrázek 12 Tvorba nového projektu

Po zvolení Create a new project (Obr. 13) treba zvoliť šablónu nového projektu a to .NET MAUI App (Preview) a kliknúť na Next, ktorý presunie na konfiguráciu projektu (Obr. 13).



Obrázek 13 Voľba šablóny projektu

Konfiguráciou nového projektu sa volí názov a umiestnenie na disku, pokračovať kliknutím tlačítka Create (Obr. 14).



Obrázek 14 Konfigurácia projektu

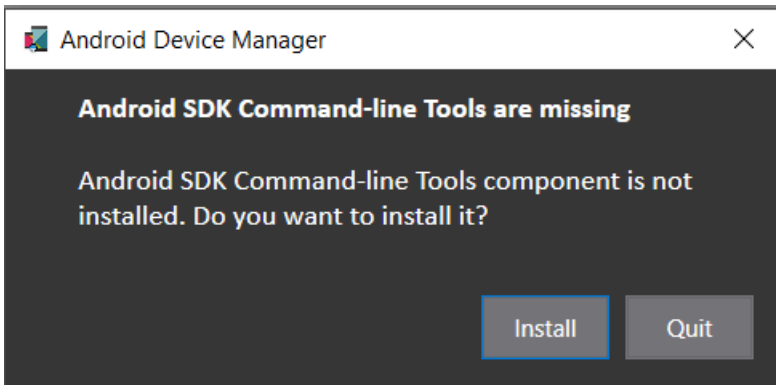
5.3 Prvé spustenie aplikácie .NET MAUI

Po úspešnom vytvorení projektu treba ešte aplikáciu spustiť na mobilnom zariadení. Je možnosť spustiť aplikáciu na reálnom zariadení alebo pomocou emulátoru vstavaného vo VS.

Prvým krokom bude spustenie debuggovania a to klikom na hornej lište na tlačítko Android Emulator (Obr. 15) alebo pomocou klávesnice F5.

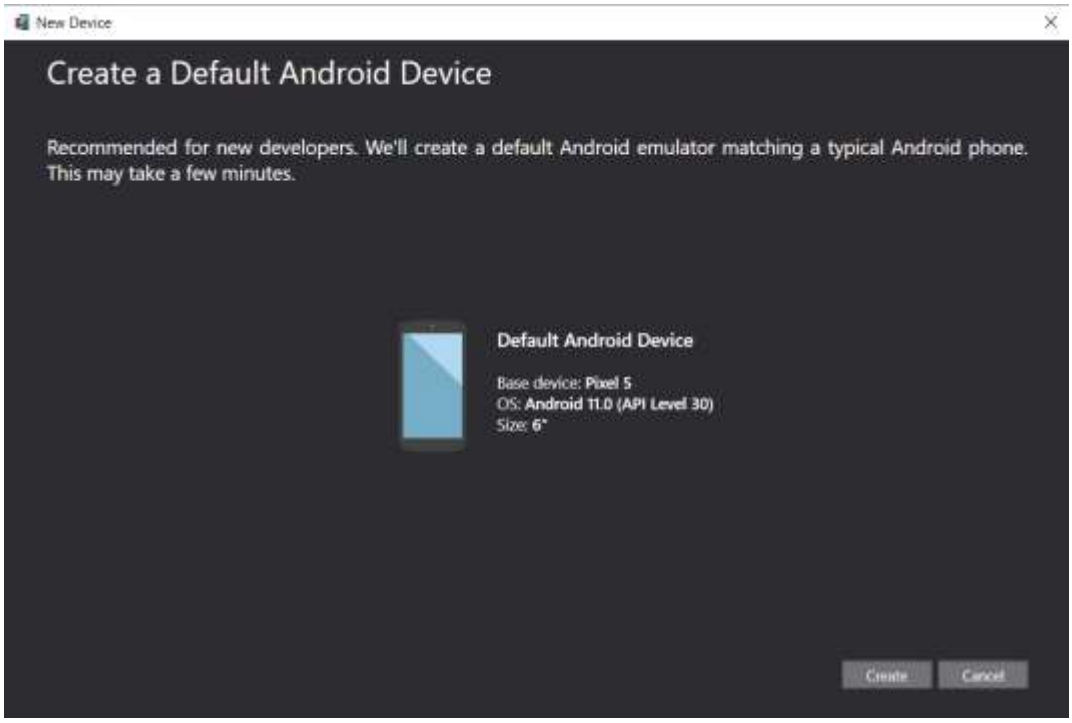
Obrázek 15 Tlačítko pre spustenie debuggovania

Po prvotnom spustený vyskočí dialógové okno Kontroly používateľských kont, potvrdiť Áno. Ďalej sa prejde k dialógovému oknu ku inštalácii Android SDK Command-line Tools, kde je treba kliknúť na Install (Obr. 16). Potvrdiť licenčné podmienky a počkať na dokončenie inštalácie.



Obrázek 16 Dialógove okno pre potvrdenie inštalácie Android SDK

Po dokončení inštalácie je potreba vytvoriť nové zariadenie, ktoré nahradí reálne zariadenie. Automaticky bude ponúknuté vytvoriť zariadenie podľa predlohy, ktorá bude vyhovovať, potvrdiť vytvorenie tlačítkom Create (Obr. 17).



Obrázek 17 Tvorba prednastaveného Android zariadenia

Spustenie emulátoru je možné urobiť 3 spôsobmi a to:

- Tools – Android – Android Device Manager (Obr. 18)
- Klávesa F5
- Lišta s tlačítkom so spusteným debuggovania (Obr. 15)

Obrázek 18 Spustenie Android emulátora

Prvé spustenie emulátora trvá dlhšie, pretože sa musí pripraviť celé virtuálne zariadenie. Po úspešnom spustení je možné pozorovať a vyskúšať si predlohu vytvorenej aplikácie.

6 NÁVRH PRAKTICKÝCH ÚLOCH VO FRAMEWORKU .NET MAUI

Vytvorilo sa 10 zadaní praktických úloh:

Úloha č. 1: Vytvoriť vstupy pre strany geometrického útvaru a následne vypísať hodnoty vypočítaného obsahu a obvodu.

Úloha č. 2: Vytvoriť tri objekty typu Radio Button, ktoré po zvolení vypíšu svoj Content.

Úloha č. 3: Vytvoriť triedu s jednou property. Táto trieda bude následne tvoriť prvky zoznamu. Bude možné voliť prvok z kolekcie. Kolekcia bude vypísaná a zmeny prvku sa musia ihneď prejaviť.

Úloha č. 4: Vytvoriť triedu s jednou property. Táto trieda bude následne tvoriť prvky zoznamu. Bude možné pridávať novú instanciu triedy do kolekcie a vymazať prvý prvok v kolekcii.

Úloha č. 5: Vytvoriť generátor hesla. Vypisovať sa bude string, ktorý bude pomocou funkcie PullToRefresh vždy náhodne vygenerovaný. Abeceda bude pozostávať z malých, veľkých písmen a číslic.

Úloha č. 6: Vytvoriť 3 objekty typu Button, ktoré spustia vždy iný druh pop-up okna. Výstup z pop-up okna sa vypíše.

Úloha č. 7: Vytvoriť kolekciu dát, ktorá bude prezentovaná pomocou CarouselView. Využiť Frame k vytvoreniu hraníc. Jedna property bude pristupovaná pomocou TwoWayBinding

Úloha č. 8: Vytvoriť vstup pre e-mail a vek, pomocou Behaviors skontrolovať vstupy, ak budú spĺňať podmienky tak text vstupu sa vypisuje zelenou farbou, v opačnom prípade červenou.

Úloha č. 9: Vytvoriť switch, ktorý po zmene stavu zmení text. Pracuje sa s rozhraním IValueConverter.

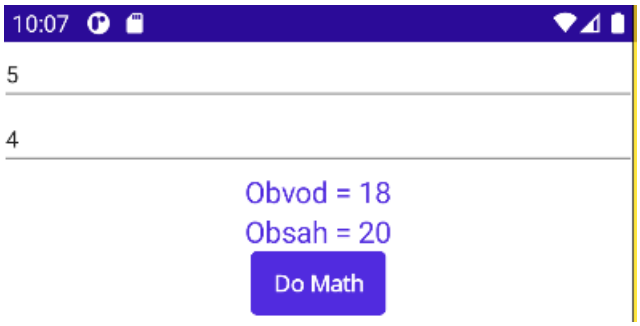
Úloha č. 10: Vytvoriť Shell aplikáciu, ktorá bude obsahovať dva krát Flyout. Obsah prvého Flyout budú dva Tab-y. Obsah druhého Flyout bude objekt Button, ktorý presmeruje užívateľa na hlavnú stránku.

Každá úloha bude obsahovať vzorové view, teoretické materiály pre vypracovanie, riešenie úlohy a kontrolné otázky.

6.1 Úloha č. 1

Zadaním úlohy bude vytvoriť vstupy pre strany geometrického útvaru a následne vypísať hodnoty vypočítaného obsahu a obvodu.

Cieľom tejto úlohy je študenta naučiť pracovať s prvkami UI ako sú Button, Label, Entry. Naučíme sa taktiež pracovať s Binding, Command a rozhraním INotifyPropertyChanged.



Obrázek 19 Vzorový View úlohy č.1

6.1.1 Materiály k teórii

StackLayout je rodičovský layout, ktorý organizuje prvky UI vertikálne a horizontálne.[44]

Entry je prvok UI, ktorý dokáže upraviť riadok textu. Je používaný k zberu malých informácii, ako napríklad heslo. [44]

Button je prvok UI, ktorý interaguje na kliknutie a vykoná Command. [44]

```
<StackLayout>
    <Entry Placeholder="Používateľské meno: " Text="{Binding Meno}"/>
    <Entry Placeholder="Heslo: " Text="{Binding Heslo}"/>
    <Button Text="Prihlásiť sa" Command="{Binding Prihlas}"/>
</StackLayout>
```

Binding slúži na jednoduché prepojenie užívateľského rozhrania a dátového modelu. [57]

BindingContext je property, ktorá špecifikuje zdrojový objekt.

```
MainPage = new MainPage() { BindingContext = new novaTrieda() };
```

Command je návrhový vzor, ktorý prevádza požiadavky alebo jednoduché operácie na objekty. [44]

```
public Command Prihlas { get; set; }

Prihlas = new Command(UserLogin);
```

Label je prvok UI, ktorý je používaný na zobrazovanie jednoriadkového alebo aj viacriadkového textu. Label sprístupňuje vlastnosť FormattedText, ktorá umožňuje prezentovať text s viacerými fontami a farbami v rovnakom zobrazení. FormattedText je typu FormattedString, ktorý obsahuje jednu alebo viac instancií Span. [44]

```
<Label Text="Ahoj"/>

<Label>
      <Label.FormattedText>
            <FormattedString >
                              <Span Text="Slon" />
                        <Span Text="Pes" />
                        </FormattedString>
      </Label.FormattedText>
</Label>
```

INotifyPropertyChanged je rozhranie, ktoré notifikuje View, že hodnota property sa zmenila. [44]

```
    internal class ViewModelBase : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;

        protected void OnPropertyChanged(string name)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
        }
    }
```

6.1.2 Riešenie úlohy

Vytvorila sa trieda GeometrickyUtvar, spracovali sa property, konštruktor a metóda aplikácie.

```
internal class GeometrickyUtvar : ViewModelBase
    {
        public double StranaA { get; set; }
        public double StranaB { get; set; }
        private double _obvod;
        private double _obsah;

        public double obvod
        {
            get { return _obvod; }
            set { _obvod = value; OnPropertyChanged(nameof(obvod)); }
        }

        public double obsah
        {
            get { return _obsah; }
            set { _obsah = value; OnPropertyChanged(nameof(obsah)); }
        }

        public Command vyrataj { get; set; }

        public GeometrickyUtvar()
        {
            vyrataj = new Command(DoMath);
        }
        private void DoMath()
        {
            obsah = StranaA * StranaB;
            obvod = 2 * StranaA + 2 * StranaB;
        }
    }
```

Následne sa prepojil dátový model na MainPage pomocou BindingContext.

```
MainPage = new MainPage() { BindingContext = new GeometrickyUtvar() }; ;
```

Vytvorila sa rodičovská trieda ViewModelBase s rozhraním INotifyPropertyChanged, ktorá pomocou eventu PropertyChanged notifikuje View o zmene hodnoty. Od tejto triedy dedí trieda GeometrickyUtvar.

```
    internal class ViewModelBase : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;

        protected void OnPropertyChanged(string name)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
        }
    }
```

Vytvoril sa layout StackLayout . Následne sa do layoutu vytvorili dva krát Entry, ktoré poslúžia ako vstupy pre property a prepojil som ich pomocou Binding. Label slúži pre výpis hodnôt, ktoré sú taktiež prepojené na dátový model pomocou Binding. Posledný bol vytvorený Button, ktorý taktiež pomocou Binding prepojíme na Command, ktorý ma priradenú metódu.

```
<StackLayout>
<Entry Placeholder="Strana A" Text="{Binding StranaA}" Keyboard="Numeric"/>
<Entry Placeholder="Strana B" Text="{Binding StranaB}" Keyboard="Numeric"/>
<Label HorizontalOptions="Center">
      <Label.FormattedText>
      <FormattedString >
            <Span Text="Obvod = " FontSize="18"/>
             <Span Text="{Binding obvod}" FontSize="18"  />
      </FormattedString>
      </Label.FormattedText>
</Label>


<Label HorizontalOptions="Center">
       <Label.FormattedText>
       <FormattedString>
             <Span Text="Obsah = " FontSize="18" />
             <Span Text="{Binding obsah}" FontSize="18"  />
       </FormattedString>
       </Label.FormattedText>
</Label>
<Button Text="Do Math" FontAttributes="Bold" Command="{Binding vyrataj}"
        HorizontalOptions="Center" />
</StackLayout>
```

6.1.3 Kontrolné otázky

Čo je to Binding?

V akom prvku UI sa využíva Command?

Na čo slúži prvok používateľského rozhrania Entry?

Ktoré rozhranie notifikuje View o zmene dát?

6.2 Úloha č. 2

Zadanie úlohy je vytvoriť tri objekty typu Radio Button, ktoré po zvolení vypíšu svoj Content.

Cieľom je naučiť študenta pracovať s RadioButton a u Label s referenciou x:Name.

Obrázek 20 Vzorový View úlohy č.2

6.2.1 Materiály k teórii

RadioButton je typ tlačítka, ktoré povoľuje užívateľovi zvoliť si jednu možnosť z balíka možností. Každá možnosť je reprezentovaná jedným RadioButtonom, defaultne každý RadioButton zobrazuje text. [44]

```
<RadioButton Content="Ronaldo"
     CheckedChanged="PlayerChanged" />

<RadioButton Content="Messi"
     CheckedChanged="PlayerChanged" />

<Label Text="Oblubeny hrac:"
     x:Name="FavoritePlayer" />
```

```
void PlayerChanged(object sender, CheckedChangedEventArgs e)
{
      RadioButton radioButton = sender as RadioButton;
      FavoritePlayer.Text = $"Oblubeny hrac: {radioButton.Content}";
}
```

Content je druh objektu, ktorý definuje string alebo view, ktorý bude zobrazený RadioButtonom. [44]

CheckedChanged je udalosť, ktorá je vyvolaná, keď sa zmení vlastnosť IsChecked. Objekt `CheckedChangedEventArgs` má jedinú hodnotu a to typu bool. [44]

x:Name identifikuje jedinečné definované XAML prvky. Dajú sa aplikovať na objekty s instanciou. Potom čo x:Name je aplikovaný na model, je názov ekvivalentný premennej, ktorá obsahuje odkaz na objekt alebo instanciu vrátenú konštruktorom. [54]

Materiály k ostatným prvkom sú opísané v kapitole 6.1.1.

6.2.2 Riešenie úlohy

Do MainPage.xaml sa vytvoril StackLayout, ktorý bude obsahovať 3 krát RadioButton a raz Label.

```
<StackLayout>
    <RadioButton Content="Pes"
         CheckedChanged="ZvieraChanged" />
    <RadioButton Content="Macka"
         CheckedChanged="ZvieraChanged" />
    <RadioButton Content="Slon"
         CheckedChanged="ZvieraChanged" />
    <Label
            Text="Zvolené zviera:"
            FontSize="18"
            FontAttributes="Bold"
            x:Name="ZvieraLabel"
            HorizontalOptions="Center" />
</StackLayout>
```

V MainPage.xaml.cs sa následne vytvorila metóda, ktorá bude mať vstupné parametre typu object a CheckedChangedEventArgs. Metóda predá parameter sender ako triedu RadioButton a vďaka tomu môže View predať radioButton.Content, ktorý sa zobrazí v Label s x:Name="ZvieraLabel".

```
partial class MainPage : ContentPage
{
      public MainPage()
      {
            InitializeComponent();
      }
      void ZvieraChanged(object sender, CheckedChangedEventArgs e)
      {
            RadioButton radioButton = sender as RadioButton;
            ZvieraLabel.Text = $"Zvolené zviera: {radioButton.Content}";
      }
}
```

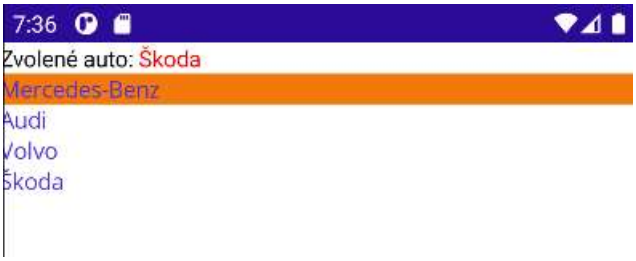
6.2.3 Kontrolné otázky

Čo je to CheckedChanged? Ako zmeníme hodnotu textu v Label pomocou C#? Prečo používame x:Name? Na čo slúži Content v RadioButton?

6.3 Úloha č. 3

Zadanie tretej úlohy bude vytvoriť triedu s jednou property. Táto trieda bude následne tvoriť prvky zoznamu. Bude možné voliť prvok z kolekcie. Kolekcia bude vypísaná a zmeny prvku sa musia ihneď prejaviť.

Cieľom tejto úlohy je študenta naučiť pracovať s Listom a s CollectionView. Využijeme taktiež prvky, s ktorými sme sa naučili pracovať v prvej úlohe, rozhranie INotifyPropertyChanged a prvok užívateľského rozhrania Label.



Obrázek 21 Vzorový View úlohy č.3

6.3.1 Materialy k teórii

CollectionView je view, ktoré prezentuje list dát používaním rozdielnych layout špecifikácii. Cieľom je poskytnúť väčšiu flexibilitu ako alternatíva ListView. Používa sa pre prezentovanie dát, ktoré požadujú scroll alebo výber. ItemSource je typ IEnumerable, špecifikuje kolekciu, ktorá sa má zobraziť defaultne je null. ItemTemplate je typ DataTemplate, špecifikuje predlohu, ktorá sa aplikuje pre každý prvok v kolekcii, ktorý má byť zobrazený. [44]

```
<CollectionView ItemsSource="{Binding Zviera}"
             <CollectionView.ItemTemplate>
                          <DataTemplate>
                                 <Label Text="{Binding Druh}" />
                          </DataTemplate>
             </CollectionView.ItemTemplate>
</CollectionView>
```

```
internal class Zviera
{
    public string Druh { get; set; }

    public Zviera(string druh)
    {
        Druh = druh;
    }
}
```

```
class Zoo : ViewModelBase
{
    private Zviera _selectedZviera;
    public List<Zviera> Zvierata { get; set; }

    public Zviera SelectedZviera
    {
        get { return _selectedZviera; }
        set { _selectedZviera = value; OnPropertyChanged(nameof(Se-
lectedZviera)); }
    }

    public Zoo()
    {
        Zvierata = new List<Zviera>
            {
                new Zviera("Slon"),
                new Zviera("Žirafa")
            };
        SelectedZviera = Zvierata.First();
    }
}
```

Materiály k ostatným potrebným prvkom sú spísane v kapitole 6.1.1.

6.3.2 Riešenie úlohy

Vytvorila sa rodičovská trieda ViewModelBase, ktorá dedí rozhranie INotifyPropertyChanged. Vytvorila sa trieda Auto, ktorá bude slúžiť ako prvok pre list.

```
    internal class Auto
    {
        public string Nazov { get; set; }
        public Auto(string nazov)
        {
            Nazov = nazov;
        }
    }
```

Vytvorila sa trieda ZoznamAut, ktorá dedí od triedy ViewModelBase. Spracovali sa property, v konštruktore sa vytvoril nový List s prvkami triedy Auto a nastavila sa property SelectedAuto na prvý prvok v liste.

```
class ZoznamAut : ViewModelBase
    {
        private Auto _selectedAuto;
        public List<Auto> Auta { get; set; }

        public Auto SelectedAuto
        {
            get { return _selectedAuto; }
            set { _selectedAuto = value; OnPropertyChanged(nameof(Selecte-
dAuto)); }
        }
```

```
        public ZoznamAut()
        {
            Auta = new List<Auto>
            {
                new Auto("Mercedes-Benz"),
                new Auto("Audi"),
                new Auto("Volvo"),
                new Auto("Škoda")
            };
            SelectedAuto = Auta.First();
        }
    }
```

V App.xaml.cs sa prepojil dátový model ZoznamAut pomocou BindingContext.

V MainPage.xaml sa vytvoril layout StackLayout. Vytvoril sa Label, ktorý bude obsahovať Label.FormattedText, v ktorom sa vytvorili dve instancie Span, prvá zobrazuje statický text a druhá bude prepojená na dátovy model pomocou Binding na property SelectedAuto. Následne sa vytvoril CollectionView, ktorý zobrazuje všetky prvky v liste a umožňuje nám meniť hodnotu property SelectedAuto v UI.

```
<StackLayout>
        <Label>
            <Label.FormattedText>
                <FormattedString>
                    <Span Text="Zvolené auto: " TextColor="Black" />
                    <Span Text="{Binding SelectedAuto.Nazov}"
                          TextColor="Red" />
                </FormattedString>
            </Label.FormattedText>
        </Label>
        <CollectionView ItemsSource="{Binding Auta}"
                        SelectedItem="{Binding SelectedAuto}"
                        SelectionMode="Single">
            <CollectionView.ItemTemplate>
                <DataTemplate>
                    <Label Text="{Binding Nazov}" />
                </DataTemplate>
            </CollectionView.ItemTemplate>
        </CollectionView>
    </StackLayout>
```

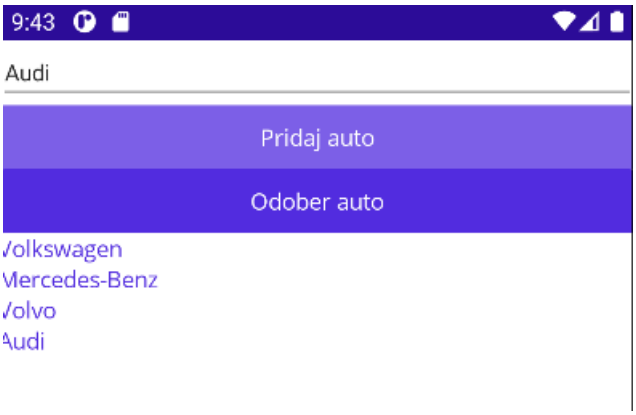
6.3.3 Kontrolné otázky

Načo slúži CollectionView? K čomu slúži SelectionMode? Ako sa vytorí list, ktorý bude obsahovať viac prvkov? Pomocou čoho sa zvolí prvý prvok v Liste

6.4 Úloha č. 4

Zadaním úlohy je vytvoriť triedu s jednou property. Táto trieda bude následne tvoriť prvky zoznamu. Bude možné pridávať novú instanciu triedy do kolekcie a vymazať prvý prvok v kolekcii.

Cieľom bude študenta naučiť pracovať s ObservableCollection. Využijeme taktiež prvky UI ako Entry, Button, Label, CollectionView, StackLayout a Command.



Obrázek 22 Vzorový View úlohy č. 4

6.4.1 Materiály k teórii

ObservableCollection reprezentuje kolekciu dát, ktorá poskytuje automatickú notifikáciu keď je pridaný nový prvok alebo prvok je odstránený, alebo napríklad keď je celá kolekcia refreshnutý. Vždy keď sa niečo takéto stane View je automaticky upravený. [44]

```
internal class Zviera
{
    public string Druh { get; set; }

    public Zviera (string druh)
    {
        Druh = druh;
    }
}
```

```
internal class Zoo
{
    public ObservableCollection<Zviera> Zvierata { get; }
    public string NovyDruh { get; set; }
    public Command PridajZviera { get; }

    public Zoo()
    {
        PridajZviera = new Command(Pridaj);
        Zvierata = new ObservableCollection<Zviera>
        {
            new Zviera("Žirafa"),
            new Zviera("Slon")
        };
    }
    private void Pridaj()
    {
        if (!string.IsNullOrEmpty(NovyDruh))
        {
            Zviera nove = new Zviera(NovyDruh);
            Zvierata.Add(nove);
        }
    }
}
```

CollectionView je opísaný v kapitole 6.3.1.

Materiály k ostatným prvkom sú opísané v kapitole 6.1.1.

6.4.2 Riešenie úlohy

Vytvorila sa trieda Auto, ktorá bude obsahovať property string Nazov.

```
internal class Auto
{
    public string Nazov { get; set; }

    public Auto(string nazov)
    {
        Nazov = nazov;
    }
}
```

Následne sa vytvorila trieda ZoznamAut, kde sa spracovali property, v konštruktore sa priradili Commandy k metódam a vytvorili sa prvky do ObservableCollection. Hodnota prvého prvku z kolekcie sa nastavila ako predvolená. Vytvorili sa metódy pre pridávanie nového prvku do kolekcie a odoberanie prvého prvku z kolekcie.

```
internal class ZoznamAut
    {
        public ObservableCollection<Auto> Auta { get; }
        public string ZnackaNovehoAuta { get; set; }
        public Command PridajAuto { get; }
        public Command OdoberAuto { get; }
        public Auto SelectedAuto;

        public ZoznamAut()
        {
            PridajAuto = new Command(Pridaj);
            OdoberAuto = new Command(Odober);
            Auta = new ObservableCollection<Auto>
            {
                new Auto("Škoda"),
                new Auto("Volkswagen"),
                new Auto("Mercedes-Benz"),
                new Auto("Volvo")
            };
            SelectedAuto = Auta.First();
        }

        private void Pridaj()
        {
            if (!string.IsNullOrEmpty(ZnackaNovehoAuta))
            {
                Auto nove = new Auto(ZnackaNovehoAuta);
                Auta.Add(nove);
            }
            SelectedAuto = Auta.First();
        }
        private void Odober()
        {
            if (Auta.Count == 1)
            {
                Auta.Remove(SelectedAuto);
            }
            if (Auta.Count >= 2)
            {
                Auta.Remove(SelectedAuto);
                SelectedAuto = Auta.First();
            }
        }
    }
```

V App.xaml.cs sa prepojil dátový model s MainPage.

```
MainPage = new MainPage() { BindingContext = new ZoznamAut() };
```

V MainPage.xaml sa vytvoril layout StackLayout, vytvoril sa prvok UI Entry, ktorý slúži ako vstup pre zadávanie nového auta. Následne sa pridali dva krát Button, ktoré sú napojené na Commandy pomocou Binding a CollectionView, ktorý zobrazí kolekciu za pomoci Label.

```
<StackLayout>
    <Entry Text="{Binding ZnackaNovehoAuta}"
           Placeholder="Značka nového auta: " Keyboard="Text"  />
    <Button Text="Pridaj auto" Command="{Binding PridajAuto}"/>
    <Button Text="Odober auto" Command="{Binding OdoberAuto}"/>
    <CollectionView ItemsSource="{Binding Auta}">
        <CollectionView.ItemTemplate>
            <DataTemplate>
                <Label Text="{Binding Nazov}" />
            </DataTemplate>
        </CollectionView.ItemTemplate>
    </CollectionView>
</StackLayout>
```

6.4.3 Kontrolné otázky

Čo to je ObservableCollection? Aký je rozdiel oproti List? Aká je metóda na pridanie nového prvku do Listu/ObservableCollection? Aká je metóda na vymazanie prvku z Listu/ObservableCollection?

6.5 Úloha č. 5

Zadanie úlohy je vytvoriť generátor hesla. Vypisovať sa bude string, ktorý bude pomocou funkcie PullToRefresh vždy náhodne vygenerovaný. Abeceda bude pozostávať z malých, veľkých písmen a číslic.

Cieľom je naučiť študenta pracovať s triedou Random a s prvkami UI RefreshView a ScrollView. Využijú sa aj prvky Label, Command a rozhranie INotifyPropertyChanged.

Obrázek 23 Vzorový View úlohy č. 5

6.5.1 Materiály k teórii

RefreshView je kontajnerový ovládač, ktorý poskytuje funkciu Pull To Refresh pre scrollovateľný obsah. Čiže potomok RefreshView musí byť scrollovateľný ako ScrollView, CollectionView alebo ListView. Command je property typu ICommand, ktorá je uskutočnená keď je refresh spustený. IsRefreshing je property typu bool, ktorá indikuje momentálny stav RefreshView. [44]

ScrollView je View, ktorý je možný scrollovať. Scroll je defaultne umožnený vertikálne. ScrollView môže mať len jedného potomka, ale môže to byť hociktorý layout. [44]

```
    <RefreshView Command="{Binding RefreshCommand}"
                 IsRefreshing="{Binding IsRefreshing}">
        <StackLayout>
            <ScrollView>
                <Label Text="{Binding Pocet}"/>
            </ScrollView>
        </StackLayout>
    </RefreshView>
```

```
class Counter : ViewModelBase{
    public Command RefreshCommand { get; set; }
    private int _pocet;
    public int Pocet { get => _pocet;
set { _pocet = value; OnPropertyChanged(nameof(Pocet)); } }
    private bool _isRefreshing;
    public bool IsRefreshing { get => _isRefreshing;
set { _isRefreshing = va lue; OnPropertyChanged(nameof(IsRefreshing)); } }

    public Counter(){
        RefreshCommand = new Command(Refresh);
        Pocet = 0;
    }
```

```
    private void Refresh(){
        ++Pocet;
        IsRefreshing = false;}}
```

Random reprezentuje pseudonáhodný číselný generátor, ktorý je algoritmus, ktorý produkuje sekvenciu čísiel. [45]

```
Random random = new Random();

int i = random.Next(0, 100);
```

Materiály k ostatným prvkom sú opísané v kapitole 6.1.1.

6.5.2 Riešenie úlohy

Vytvorila sa trieda RandomString a rodičovská trieda ViewModelBase. V triede RandomString sa spracovali property a globálna premenná triedy Random. V konštruktore sa predala commandu metóda a spracovala hodnota property randomString. Vytvorila sa metóda Refresh, ktorá vygeneruje náhodný string z danej abecedy znakov o náhodne generovanej dĺžke.

```
internal class RandomString : ViewModelBase
    {
        public Command RefreshCommand { get; set; }
        private string _randomString;
        public string randomString { get => _randomString; set { _randomString
= value; OnPropertyChanged(nameof(randomString)); } }
        private bool _isRefreshing;
        public bool IsRefreshing { get => _isRefreshing; set { _isRefreshing =
value; OnPropertyChanged(nameof(IsRefreshing)); } }
        Random random = new Random();

        public RandomString()
        {
            RefreshCommand = new Command(Refresh);
            randomString = "";
        }

        private void Refresh()
        {
            string abeceda = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstu-
vwxyz0123456789";
            int l = random.Next(5, 20);
            var stringChars = new char[l];
            for (int i = 0; i < l; i++)
            {
                stringChars[i] = abeceda[random.Next(abeceda.Length + 1)];
            }
            randomString = new String(stringChars);
            IsRefreshing = false;
        }
    }
```

Vytvoril sa RefreshView, ktorému sa pomocou Binding priradil command a premenná typu bool. Layout StackLayout obsahuje prvok UI Label, ktorý bude mať statický text a ScrollView, ktorý umožňuje scrollovať a využiť funkciu Pull To Refresh pre spustenie metódy na generáciu náhodného stringu a jeho hodnotu zobrazí pomocou Label.

```
<RefreshView Command="{Binding RefreshCommand}"
             IsRefreshing="{Binding IsRefreshing}"
             HorizontalOptions="Center">
    <StackLayout>
        <Label Text="Tvoj náhodny string:" FontSize="30" />
        <ScrollView>
            <Label Text="{Binding randomString}"
                   FontSize="20" HorizontalOptions="Center" />
        </ScrollView>
    </StackLayout>
</RefreshView>
```

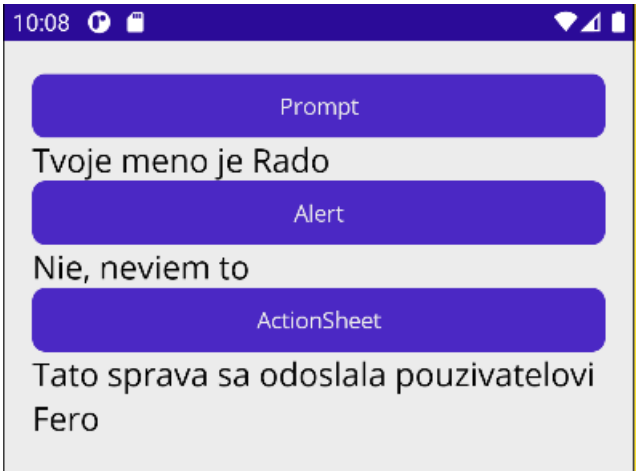
6.5.3 Kontrolné otázky

Aká funkcia je využiteľná vďaka RefreshView? Aká metóda triedy Random nám vracia náhodný integer? K čomu slúži ScrollView?

6.6 Úloha č. 6

Zadaním úlohy Vytvoriť 3 objekty typu Button, ktoré spustia vždy iný druh pop-up okna. Výstup z pop-up okna sa vypíše.

Cieľom tejto úlohy je študenta naučiť pracovať s DisplayPromptAsync, DisplayAlert a DisplayActionSheet. Taktiež práca s async a await. Využité budú Label a Button.



Obrázek 24 Vzorový View úlohy č. 6

6.6.1 Materiály k teórii

Async sa používa na špecifikovanie metódy, že je asynchrónna. Môže teda bežať zarovno ďalšieho tasku bez vzájomného prerušenia.[47]

Await operátor používame pred asynchrónnym volaním, keď chceme aby sa ďalší beh programu umiestený za await naplánoval a spustil až po dokončení operácie. Každá metóda, ktorá využíva await musí byť asynchrónna. Neblokuje volajúce vlákno.[47]

DisplayActionSheet ponúka užívateľovi sadu alternatívnych možností. Potom čo užívateľ klikne na jedno z tlačítiek sa výstup vráti ako string. Podporuje taktiež možnosť destroy, ktorá má deštruktívne chovanie. Je to tretí argument alebo môže byť ponechaný null [48]:

```
async void OnActionSheetCancelDeleteClicked(object sender, EventArgs e)
{
    string action = await DisplayActionSheet("ActionSheet: SavePhoto?", "Can-
cel", "Delete", "Photo Roll", "Email");
    Debug.WriteLine("Action: " + action);
}
```

DisplayAlert je modálne pop-up okno, ktoré užívateľa upozorní alebo mu položí jednoduchú otázku [48]:

```
await DisplayAlert("Alert", "You have been alerted", "OK");
```

Metódu DisplayAlert sa dá taktiež použiť k zachyteniu odozvy užívateľa predložením dvoch tlačítiek a vrátením bool [48]:

```
bool answer = await DisplayAlert("Question?", "Would you like to play
a game?", "Yes", "No");
Console.WriteLine("Answer: " + answer);
```

DisplayPrompt je alert, ktorý zobrazuje výzvu, napríklad odpovedať na otázku. Po stlačení OK sa zadaná odpoveď predá ako string, pokiaľ sa stlačí Cancel vráti sa null [48]:

```
string result = await DisplayPromptAsync("Question 1", "What's your name?");
```

Materiály k ostatným prvkom sú opísané v kapitole 6.1.1.

6.6.2 Riešenie úlohy

V MainPage.xaml sa vytvoril StackLayout, do ktorého sa následne vytvoril 3x Button a 3x Label. Každý button slúži pre spustenie jedného druhu alertu. Ku každému Buttonu je priradený jeden Label, ktorý vypíše vždy výstup z priradeného Alertu.

```
<StackLayout Margin="20">
        <Button Text="Prompt"
                Clicked="PromptClicked" />
        <Label FontSize="20"
               Text=""
               x:Name="PromptLabel" />

        <Button Text="Alert"
                Clicked="AlertClicked"/>
        <Label FontSize="20"
               Text=""
               x:Name="AlertLabel" />

        <Button Text="ActionSheet"
                Clicked="ActionSheetClicked"/>
        <Label FontSize="20"
               Text=""
               x:Name="ActionSheetLabel" />
    </StackLayout>
```

V triede MainPage.xaml.cs sa vytvorili 3 metódy zhodujúce sa s názvom Button.Clicked. Každá metóda spustí iný druh alertu a následne vypíše výstup do Label.

```
async void PromptClicked(object sender, EventArgs e)
{
	string meno = await DisplayPromptAsync("Prompt", "Ako sa volas?");
	PromptLabel.Text = "Tvoje meno je " + meno;
}

async void AlertClicked(object sender, EventArgs e)
{
	bool odpoved = await DisplayAlert("Alert","Rad programujes?","Ano",
"Nie");
	if(odpoved == true)
	{
		AlertLabel.Text = "Ano, rad programujem";
	}
	else
	{
		AlertLabel.Text = "Nie, neviem to";
	}
}

async void ActionSheetClicked(object sender, EventArgs e)
{
	string kontakt = await DisplayActionSheet("ActionSheet", "Zrus", null,
"Jan", "Fero", "Lukas");
	ActionSheetLabel.Text = "Tato sprava sa odoslala pouzivatelovi " + kon-
takt;
}
```

6.6.3 Kontrolné otázky

K čomu sa dá použiť DisplayPromptAsync?

K čomu slúži DisplayAlert?

Aké využitie môže mať DisplayActionSheet?

6.7 Úloha č. 7

Vytvoriť kolekciu dát, ktorá bude prezentovaná pomocou CarouselView. Využiť Frame k vytvoreniu hraníc. Jedna property bude pristupovaná pomocou TwoWayBinding Cieľom tejto úlohy je študenta naučiť pracovať s CollectionView, TwoWayBinding a Frame.

Obrázek 25 Vzorový View úlohy č. 7

6.7.1 Materiály k teórii

CarouselView je view, ktorý reprezentuje dáta v skrolovaciom layoute, kde môže používateľ posúvaním sa pohybovať medzi položkami v kolekcii. Obecne zobrazuje položky horizontálne orientované. Jedna položka v liste je vždy zobrazená na obrazovke. Taktiež môže byť zobrazený indikátor na ktorom indexe v liste sa nachádzame. Poskytuje slučkový prístup, čiže ak na poslednom indexe posunieme ďalej, dostaneme sa na prvý index a naopak, z prvého posunieme dozadu tak sa dostaneme na posledný.[44]

```
internal class Zviera

{
        public string Druh { get; set; }

        public Zviera (string druh)
        {
            Druh = druh;
        }
}

class Zvierata
{
        public ObservableCollection<Zviera> Zoo { get; set; }

        public Zvierata ()
        {
            Zoo = new ObservableCollection< Zviera >
            {
                new Zviera("Slon"),
                new Zviera ("Jazvec"),
                new Zviera ("Tiger")
            };
}}
```

```
<CarouselView ItemsSource="{Binding Auta}">
    <CarouselView.ItemTemplate>
        <DataTemplate>
            <StackLayout>
                <Label Text="{Binding Znacka}"
                       HorizontalOptions="Center"/>
            </StackLayout>
        </DataTemplate>
    </CarouselView.ItemTemplate>
</CarouselView>
```

TwoWayBinding je viazanie dát, ktoré spôsobuje že zmeny v zdrojovom property alebo v cieľovom property as automaticky aktualizujú. Tento typ väzby je vhodný pre interaktívne scenáre UI.[56][57]

```
<Label Text="{Binding Path=Druh, StringFormat=Druh zvierata: {0}'}"
                                  HorizontalOptions="Center" />
```

Frame je používaný k zabaleniu View alebo layout s hranicami, ktoré môžu byť nastavené farbou, tieňmi a ďalšími možnosťami [49]:

```
<Frame BorderColor="Gray"
       CornerRadius="10">
      <Label Text="Frame okolo Label" />
</Frame> [49]
```

Materiály k ostatným potrebným prvkom sú opísané v kapitole 6.1.1.

6.7.2 Riešenie úlohy

Vytvorila sa trieda Auto, ktorá obsahuje štyri property, ktoré budú tvoriť detaily kartičky v UI.

```
internal class Auto
{
    public string Znacka { get; set; }
    public string Typ { get; set; }
    public double Objem { get; set; }
    public int RokVyroby { get; set; }

    public Auto(string znacka, string typ, double objem, int rokVyroby)
    {
        Znacka = znacka;
        Typ = typ;
        Objem = objem;
        RokVyroby = rokVyroby;
    }
}
```

Vytvorila sa trieda ZoznamAut, kde sa spracovala kolekcia instancií triedy Auta.

```
class ZoznamAut
    {
        public ObservableCollection<Auto> Auta { get; set; }

        public ZoznamAut()
        {
            Auta = new ObservableCollection<Auto>
            {
                new Auto("Mercedes-Benz","E" ,2.2, 2018),
                new Auto("Audi","A6",3.0, 2016),
                new Auto("Skoda","Octavia",1.6, 2021)
            };
        }}}
```

Následne sa tvorilo UI v MainPage.xaml, vytvoril sa CarouselView, následne sa pod DataTemplate pridal layout StackLayout, pod ktorý sa vytvoril Frame, ktorý v UI zobrazí kartičku s detailmi. Pod Frame sa znova pridal StackLayout, ktorý bude obsahovať 4x Label, ktoré sú pomocou Binding napojené na dátový model a danú property. Jeden Label sa napojil na dátový model pomocou TwoWayBinding a to pre zobrazovanie property objem.

```
<CarouselView ItemsSource="{Binding Auta}">
        <CarouselView.ItemTemplate>
            <DataTemplate>
                <StackLayout>
                    <Frame BorderColor="Red"
                       CornerRadius="10"
                       HeightRequest="120"
                       HorizontalOptions="Center">
                        <StackLayout>
                            <Label Text="{Binding Znacka}"
                               FontAttributes="Bold"
                               FontSize="18"
                               HorizontalOptions="Center" />
                            <Label Text="{Binding Typ}"
                               HorizontalOptions="Center" />
                            <Label Text="{Binding RokVyroby}"
                               HorizontalOptions="Center" />
                            <Label Text="{Binding Path=Objem,
                                   StringFormat='Objem: {0:F1} l'}"
                                   HorizontalOptions="Center" />
                        </StackLayout>
                    </Frame>
                </StackLayout>
            </DataTemplate>
        </CarouselView.ItemTemplate>
    </CarouselView>
```

6.7.3 Kontrolné otázky

K čomu slúži CarouselView? Aký prístup má CarouselView? Aký dôvod má využitie TwoWayBinding? K čomu slúži Frame.CornerRadius?

6.8 Úloha č. 8

Zadaním úlohy je vytvoriť vstup pre e-mail a vek, pomocou Behaviors skontrolovať vstupy, ak budú spĺňať podmienky tak text vstupu sa vypisuje zelenou farbou, v opačnom prípade červenou. Cieľom je naučiť študenta pracovať s Behaviors. Použitý bude UI prvok Entry.

Obrázek 26 Vzorový View úlohy č. 8

6.8.1 Materiály k teórii

Behaviors dovoľujú pridať funkcionalitu do UI bez toho aby sa im vytvárali dedičné triedy. Miesto toho je funkcionalita implementovaná v Behavior triede a priradená k prvku UI ako keby bola jeho súčasťou. .NET MAUI behaviors sú vytvorené odvodením z triedy Behavior alebo Behavior<T>[50]:

```
public class MyBehavior : Behavior<Entry>
{
    protected override void OnAttachedTo(Entry bindable)
    {
        // Perform setup
        bindable.TextChanged += ChangedTextBehavior;
        base.OnAttachedTo(bindable);
    }

    protected override void OnDetachingFrom(Entry bindable)
    {
        // Perform clean up
        bindable.TextChanged -= ChangedTextBehavior;
        base.OnDetachingFrom(bindable);
    }

    void ChangedTextBehavior(object sender, TextChangedEventArgs args)
    {
        // Behavior implementacia
    }
}
```

```
<Entry Placeholder="Enter text">
    <Entry.Behaviors>
        <local:MyBehavior />
    </Entry.Behaviors>
</Entry>
```

Materiály k Label sú opísané v kapitole 6.1.1.

6.8.2 Riešenie úlohy

Vytvorila sa trieda AgeValidationBehavior, ktorá bude dediť od triedy Behavior, kde Entry bude objekt. Následne sa preťažili zdedené metódy pre zmenu textu. Metóda vyhodnocuje vstupnú hodnotu a keď je zadaný vek väčší alebo rovný 18, tak zobrazí text zelenou farbou, inak bude zobrazovať text červenou.

```
public class AgeValidationBehavior : Behavior<Entry>
    {
        protected override void OnAttachedTo(Entry entry)
        {
            entry.TextChanged += EntryAge;
            base.OnAttachedTo(entry);
        }

        protected override void OnDetachingFrom(Entry entry)
        {
            entry.TextChanged -= EntryAge;
            base.OnDetachingFrom(entry);
        }

        void EntryAge(object sender, TextChangedEventArgs args)
        {
            double result;
            Double.TryParse(args.NewTextValue, out result);
            ((Entry)sender).TextColor = result >= 18 ? Colors.Green : Co-
lors.Red;
        }
    }
```

Vytvorila sa trieda EmailValidationBehavior, ktorá bude dediť od triedy Behavior, kde Entry bude objekt. Následne sa preťažili zdedené metódy pre zmenu textu. Metóda vyhodnocuje vstupnú hodnotu stringu, pokiaľ neobsahuje @ tak bude bool false a teda text červenou farbou, ak obsahuje bool bude true a text zelenou farbou.

```
public class EmailValidationBehavior : Behavior<Entry>{
        protected override void OnAttachedTo(Entry entry){
            entry.TextChanged += EntryEmail;
            base.OnAttachedTo(entry);
        }

        protected override void OnDetachingFrom(Entry entry){
            entry.TextChanged -= EntryEmail;
            base.OnDetachingFrom(entry);
        }

        void EntryEmail(object sender, TextChangedEventArgs args){
            bool isValid;
            if (!args.NewTextValue.Contains("@")){
                isValid = false;
            }
            else{
                isValid = true;
            }
            ((Entry)sender).TextColor = isValid ? Colors.Green : Colors.Red;
        }}
```

V MainPage.xaml sa vytvorili dva vstupy Entry, kde sa pridali Entry.Behaviors. Na začiatok sa pridal riadok `xmlns:local`, aby bolo umožnené triedam AgeValidationBehavior a EmailValidationBehavior fungovať.

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:Uloha8"
             x:Class="Uloha8.MainPage">
    <StackLayout>
        <Entry Placeholder="Zadaj vek">
            <Entry.Behaviors>
                <local:AgeValidationBehavior />
            </Entry.Behaviors>
        </Entry>
        <Entry Placeholder="Zadaj e-mail">
            <Entry.Behaviors>
                <local:EmailValidationBehavior />
            </Entry.Behaviors>
        </Entry>
    </StackLayout>
</ContentPage>
```

6.8.3 Kontrolné otázky

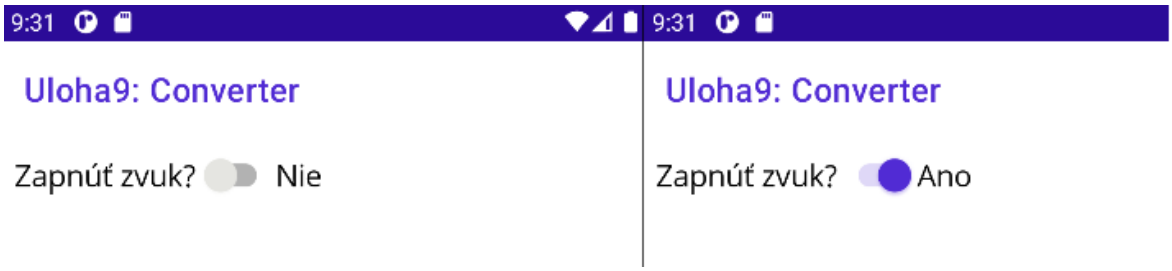
V Behavior<T> čo môžeme dosadiť za T?

Aký modifikátor metóda musí mať keď ju preťažíme pomocou override?

6.9 Úloha č. 9

Zadaním úlohy je vytvoriť switch, ktorý po zmene stavu zmení text. Pracuje sa s rozhraním IValueConverter.

Cieľom je naučiť pracovať so Switch, x: referenciami a s rozhraním IValueConverter.



Obrázek 27 Vzorový View úlohy č. 9

6.9.1 Materiály k teórii

Dátové väzby obvykle prenášajú dáta zo zdrojovej property do cieľovej property. Tento prenos je priamočiary, ak sú property rovnakého typu, alebo ak sa dá jeden typ previesť na druhý pomocou implicitnej konverzie. Ak tomu tak nie je musí prebehnúť typová konverzia. Pre iné typy konverzií musíte napísať kód v triede, ktorá implementuje rozhrania IValueConverter. Triedy, ktoré implementujú toto rozhranie sa nazývajú prevodníky hodnôt alebo taktiež prevodníky väzieb.[51]

Trieda prevodníka hodnôt môže mať property a generické parametre. Nasledujúci prevodník hodnôt konvertuje bool na objekt typu T: [51]

```
public class BoolToObjectConverter<T> : IValueConverter
{
    public T TrueObject { get; set; }
    public T FalseObject { get; set; }

    public object Convert(object value, Type targetType, object parameter,
CultureInfo culture)
    {
        return (bool)value ? TrueObject : FalseObject;
    }

    public object ConvertBack(object value, Type targetType, object parameter,
CultureInfo culture)
    {
        return ((T)value).Equals(TrueObject);
    }
}
```

Nasledujúci príklad ukazuje, ako tento konvertor použiť na zobrazenie hodnoty Switch. Tu sa prevodník vytvorí medzi tagmi Binding.Converter. x:TypeArguments označuje generický argument a TrueObject a FalseObject sú nastavené ako objekty tohto typu: [51]

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:DataBindingDemos"
             x:Class="DataBindingDemos.SwitchIndicatorsPage"
             Title="Switch Indicators">
    <ContentPage.Resources>
        <Style TargetType="Label">
            <Setter Property="FontSize" Value="18" />
            <Setter Property="VerticalOptions" Value="Center" />
        </Style>

        <Style TargetType="Switch">
            <Setter Property="VerticalOptions" Value="Center" />
        </Style>
    </ContentPage.Resources>
    <StackLayout Padding="10, 0">
        <StackLayout Orientation="Horizontal"
                     VerticalOptions="Center">
            <Label Text="Allow popups?" />
            <Switch x:Name="switch" />
            <Label>
                <Label.Text>
                    <Binding Source="{x:Reference switch}"
                             Path="IsToggled">
                        <Binding.Converter>
                            <local:BoolToObjectConverter
                                                x:TypeArguments="x:String"
                                                TrueObject="Yes"
                                                FalseObject="No" />
                        </Binding.Converter>
                    </Binding>
                </Label.Text>
                <Label.TextColor>
                    <Binding Source="{x:Reference switch}"
                             Path="IsToggled">
                        <Binding.Converter>
                            <local:BoolToObjectConverter
                                                x:TypeArguments="Color"
                                                TrueObject="Green"
                                                FalseObject="Red" />
</Binding.Converter></Binding></Label.TextColor></Label></StackLay-
out></StackLayout></ContentPage>
```

Switch je prvok UI predstavujúci horizontálny prepínač, ktorý môže byť manipulovaný užívateľom a prepínaný medzi stavmi, ktoré reprezentujú boolean hodnotu.[53]

Často je potrebné vytvárať instancie objektov pomocou konštruktorov, ktoré vyžadujú argumenty, alebo volaním metódy statického vytvorenia. To sa dá dosiahnuť pomocou XAML atribútov x:Arguments. Atribút x:Arguments sa používa na špecifikovanie argumentov konštruktora pre iný ako predvolený konštruktor alebo pre deklaráciu objektu.[52]

Materiály k prvku Label sú opísané v kapitole 6.1.1.

6.9.2 Riešenie úlohy

Vytvorila sa trieda BoolToStringConverter, ktorá dedí od rozhrania IValueConverter. Vytvorili sa 2 property, ktoré pracujú ako premenné pre bool true a false. Metódy z rozhrania pracujú pre konverziu z bool na string a spätnú konverziu zo stringu na bool.

```
    internal class BoolToStringConverter<T> : IValueConverter
    {
        public T True { get; set; }
        public T False { get; set; }

        public object Convert(object value, Type targetType, object parameter,
CultureInfo culture)
        {
            if ((bool)value) return "Ano";
            else return "Nie";
        }

        public object ConvertBack(object value, Type targetType, object para-
meter, CultureInfo culture)
        {
            if ((string)value == "Ano") return true;
            else return false;
        }
```

V MainPage.xaml sa vytvoril rodičovský layout a layout v ktorom sa pridal dva krát Label a Switch. Prvý Label obsahuje statický text. Switch má referenciu x:Name, ktorá predáva aktuálny stav v bool. Nasledujúci Label je dynamický, pretože využíva Binding na triedu BoolToStringConverter a referenciu na Switch, čiže ak je switch v stave false tak vypíše Nie a ak je v stave true vypíše sa Áno.

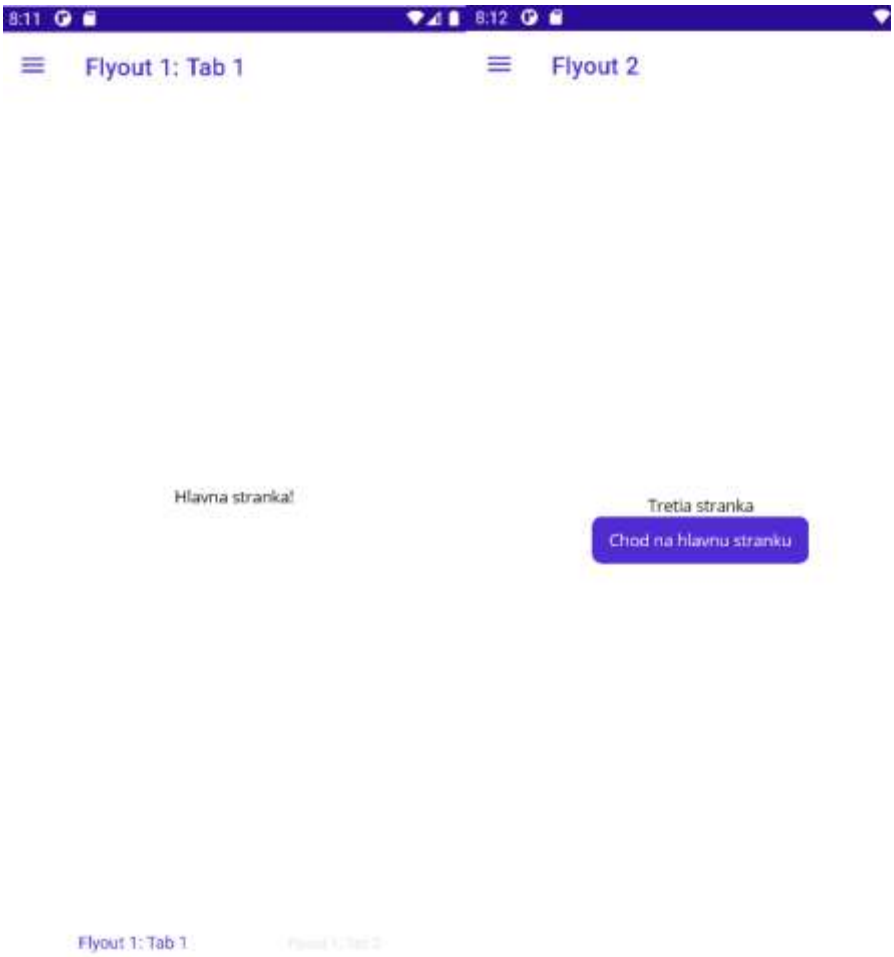
```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:convert="clr-namespace:Uloha9"
             x:Class="Uloha9.MainPage">

    <StackLayout Padding="10, 0">
        <StackLayout Orientation="Horizontal"
                     VerticalOptions="Center">
            <Label Text="Zapnúť zvuk?" VerticalOptions="Center"
FontSize="18"/>
            <Switch x:Name="Prepinac" VerticalOptions="Center"/>
            <Label VerticalOptions="Center" FontSize="18">
                <Label.Text>
                    <Binding Source="{x:Reference Prepinac}"
                             Path="IsToggled">
                        <Binding.Converter>
                            <convert:BoolToStringConverter
                                        x:TypeArguments="x:String" />
                        </Binding.Converter></Binding>
        </Label.Text></Label></StackLayout></StackLayout>
</ContentPage>
```

6.9.3 Kontrolné otázky

Aký návratový typ je switch? K čomu slúži rozhranie IValueConverter?

6.10 Úloha č. 10

Zadaním úlohy je vytvoriť Shell aplikáciu, ktorá bude obsahovať dva krát Flyout. Obsah prvého Flyout budú dva Tab-y. Obsah druhého Flyout bude objekt Button, ktorý presmeruje užívateľa na hlavnú stránku. Cieľom úlohy je naučiť študenta pracovať so Shell. Využité budú prvky UI Command, Label a Button.



Obrázek 28 Vzorový View úlohy č. 10

6.10.1 Materiály k teórii

Shell redukuje komplexitu vývoja aplikácie poskytovaním základných funkcií, ktoré väčšina aplikácii požaduje. Vizuálna hierarchia aplikácie môže pozostávať z 3 objektov. [46]

FlyoutItem reprezentuje jeden alebo viac položiek vo flyout a mal by byť použitý ak navigačný vzor vyžaduje flyout. TabBar reprezentuje spodný tab bar a mal by byť použitý ak navigačný vzor aplikácie začína so spodnými tabmi a nepotrebuje flyout. Každý FlyoutItem alebo TabBar objekt je potomok Shell objektu.[55]

Tab, ktorý reprezentuje zoskupený obsah navigovaný spodnými tabmi. Každý Tab objekt je potomok FlyoutItema lebo TabBar. [55]

ShellContent je objekt, ktorý reprezentuje ContentPage pre každý tab. Každý ShellContent je potomok Tab objektu. Ak je viac potomkov v Tab objekte, objekty budú navigované vrchnými tabmi [55]:

```
<Shell xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
       xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
       xmlns:local="clr-namespace:Uloha10"
       x:Class="Uloha10.AppShell">
    ...
    <FlyoutItem Title="Flyout">
        <Tab Title="Spodny Tab1">
            <ShellContent Title="Horny Tab 1"
                          ContentTemplate="{DataTemplate local:MainPage}"/>
            <ShellContent Title="Horny Tab 2"
                          ContentTemplate="{DataTemplate local:MainPage}"/>
        </Tab>

        <ShellContent Title="Spodny Tab 2"
                      ContentTemplate="{DataTemplate local:SecondPage}" />
    </FlyoutItem>
    ...
</Shell>
```

Shell má implicitnú konverziu, ktorá povoľuje Shell hierarchii byť zjednodušená. Toto je možné preto, že potomkovia Shell môžu obsahovať len FlyoutItem alebo TabBar, ktoré môžu obsahovať len Tab objekty, a tie zasa len ShellContent. Táto implicitná konverzia automaticky zabalí ShellContent objekt pod Tab objekt. [55]

Navigácia je uskutočnená v Shell špecifikáciou URL kam sa má navigovať. Route automaticky vytvára a registruje cestu. Tvorí sa nasledujúca hierarchia [46]:

```
<Shell ...>
    <FlyoutItem ...
                Route="zvierata">
        <Tab ...
             Route="domace">
            <ShellContent ...
                          Route="macky" />
        </Tab>
        <ShellContent ...
                      Route="kravy" />
    </FlyoutItem>
    ...
</Shell>
```

K presmerovaniu do ShellContent objektu pre cestu macky je absolutna URL //zvierata/domace/macky. [46]

Materiály k ostatným prvkom sú opísané v kapitole 6.1.1.

6.10.2 Riešenie úlohy

V AppShell.xaml sa vytvorila hierarchia navigácii. ContentTemplate predstavuje, ktorý ContentPage bude k tomu objektu priradený. Route automaticky vytvorí a zaregistruje požadovanú cestu.

```
    <FlyoutItem Title="Flyout 1">
        <ShellContent   Title="Flyout 1: Tab 1" ContentTemplate="{DataTemplate
local:MainPage}" Route="MainPage" />
        <ShellContent   Title="Flyout 1: Tab 2" ContentTemplate="{DataTemplate
local:SecondPage}"   Route="SecondPage" />
    </FlyoutItem>
    <FlyoutItem Title="Flyout 2">
        <ShellContent   Title="Flyout 2" ContentTemplate="{DataTemplate
local:ThirdPage}"   Route="ThirdPage" />
    </FlyoutItem>
```

Vytvorilo sa UI stránky MainPage.xaml.

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Uloha10.MainPage">

    <Label Text="Hlavna stranka!" VerticalOptions="Center" HorizontalOp-
tions="Center"/>
</ContentPage>
```

Následne sa vytvorilo UI stránky SecondPage.xaml.

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Uloha10.SecondPage">

    <Label Text="Druha stranka!" VerticalOptions="Center" HorizontalOp-
tions="Center"/>
</ContentPage>
```

Posledným krokom tvorby UI sa vytvorila ThirdPage.xaml.

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Uloha10.ThirdPage">

    <StackLayout HorizontalOptions="Center" VerticalOptions="Center">
        <Label Text="Tretia stranka" HorizontalOptions="Center" />
        <Button Text="Chod na hlavnu stranku"
                Command="{Binding GotoMainPageCommand}" />
    </StackLayout>
</ContentPage>
```

Následne sa vytvorila trieda ViewModel, ktorá obsahuje Command a asynchrónnu metódu, ktorá presmeruje na MainPage.

```
public Command GotoMainPageCommand { get; set; }

public ViewModel()
{
    GotoMainPageCommand = new Command(GotoMainPage);
}
private async void GotoMainPage()
{
    await Shell.Current.GoToAsync("//MainPage");
}
```

Posledným krokom bolo v súbore App.xaml.cs prepojenie triedy ViewModel pomocou BindingContext na MainPage.

```
MainPage = new AppShell() { BindingContext = new ViewModel() };
```

6.10.3 Kontrolné otázky

Aká je hierarchia Shell? Ak obsahuje Tab viac krát potomkov ShellContent ako sa to prejaví? ShellContent môže byť potomkom len akého objektu?

ZÁVĚR

Cieľom bakalárskej práce bolo popísanie a demonštrácia využitia frameworku .NET MAUI. Práca sa zaoberala najmä technológiami multiplatformového vývoja mobilných aplikácii. Súčasťou práce je taktiež sada výučbových materiálov, čo znamená krátke úlohy na precvičenie pre študentov.

V prvej kapitole práca objasňuje čo to je mobilné zariadenie. V tejto kapitole sú spomenuté najpoužívanejšie operačné systémy, a to Android a iOS. Ku každému operačnému systému bol pridaný popis ich architektúr. Práca poskytuje aj pohľad na súčasný vývoj mobilných aplikácii, či už sa jedná o natívne aplikácie alebo multiplatformové, ku ktorým patria aj hybridné a webové aplikácie. Okrem popisu hybridných a webových aplikácii sa ku každému typu napísal krátky súhrn výhod a ich nevýhod.

Ďalej práca obsahuje súčasné technológie pre vývoj multiplatformových aplikácií. V tejto kapitole sa popísali technológie, ktoré momentálne patria medzi najznámejšie a pravdepodobne aj najpoužívanejšie. Patria tam technológie ako .NET, Xamarin.Forms alebo React Native, ak by niekoho zaujímalo viac o týchto technológiách, sú odkazované priamo na ich oficiálne dokumentácie. V neposlednom rade je vlastná kapitola venovaná technológii .NET MAUI, kde je popísané ako pracuje, architektúra a podporované platformy.

Prvá kapitola praktickej časti bola venovaná príprave prostredia pre prácu s .NET MAUI. V tejto kapitole som dopodrobna vysvetlil postup inštalácie, prvé vytvorenie a spustenie aplikácie na mobilné zariadenie, emulátor vo Visual Studio 2022.

Posledná kapitola v praktickej časti už bola samotná tvorba zadaní úloh a teoretických materiálov pre študentov. Samotných úloh je 10. Prostredníctvom vypracovaných úloh si študent prejde základmi tvorby aplikácie v .NET MAUI po zložitejšie funkcie a navrhovanie UI. Napríklad si študent precvičí viazanie dát, tvorbu vyskakovacích okien alebo carousel view. Taktiež sa naučia pracovať s rozhraniami. Každá úloha teda obsahuje aj samotné riešenie a kontrolné otázky, ktoré by dotyčný učiteľ mohol využiť.

SEZNAM POUŽITÉ LITERATURY

[1] PETZOLD, CHARLES. *Creating Mobile Apps with Xamarin.Forms: Cross-platform C# programming for iOS, Android, and Windows*. Redmond, Washington: Microsoft Press, c2016. ISBN 978-1-5093-0297-0.

[2] IVERSEN, Jakob a Michael EIERMAN, ©2014. *Learning Mobile App Development: A Hands-on Guide to Building Apps with iOS and Android*. Crawfordsville (Indiana): Pearson Education. ISBN 978-0-321-94786-4.

[3] Mobile Operating System Market Share Worldwide, c1999-2022. *Statcounter: Global Stats* [online]. Dublin: StatCounter [cit. 2022-02-17]. Dostupné z: https://gs.statcounter.com/os-market-share/mobile/worldwide

[4] What is Android, ©2011-2021. *Javatpoint* [online]. JavaTpoint [cit. 2022-03-01]. Dostupné z: https://www.javatpoint.com/android-what-where-and-why

[5] KARCH, Marziah. What is Android: The ins and outs of the Android operating system. *Lifewire* [online]. Dotdash [cit. 2022-03-01]. Dostupné z: https://www.lifewire.com/what-is-google-android-1616887

[6] Platform Architecture. *Android Developers* [online]. [cit. 2022-03-01]. Dostupné z: https://developer.android.com/guide/platform

[7] Mobile Application Development: What is mobile application development?. *IBM Cloud Learn Hub* [online]. [cit. 2022-03-01]. Dostupné z: https://www.ibm.com/cloud/learn/mobile-application-development-explained

[8] PRICE, Mark J., © 2020. *C# 9 and .NET 5: Modern Cross-Platform Development*. 5th Edition. Birmingham: Packt Publishing. ISBN 978-1-80056-810-5.

[9] Cross-Platform Mobile Development: Five Best Frameworks. *SaM Solutions* [online]. Gilching: SaM Solutions, © 1993-2021, 25.10.2019 [cit. 2022-02-14]. Dostupné z: https://www.sam-solutions.com/blog/cross-platform-mobile-development/

[10] Cross-Platform App Development – Explore Frameworks, Technology and Business Benefits. *Asper Brothers* [online]. Warsaw: ©ASPER BROTHERS, ©2022, 13.12.2020 [cit. 2022-02-14]. Dostupné z: https://asperbrothers.com/blog/cross-platform-app-development/

[11] OLSON, Scott, © 2012. *Professional Cross-Platform Mobile Development in C#*. Indianapolis: John Wiley. ISBN 978-1-118-15770-1.

[12] What is .NET MAUI?, © 2022. *Microsoft technical documentation* [online]. Redmont: Microsoft [cit. 2022-02-17]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/maui/what-is-maui

[13] Mobile development with .NET, © 2022. *Microsoft technical documentation: .NET MAUI Installation* [online]. Redmont: Microsoft [cit. 2022-02-17]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/maui/get-started/media/installation/vs-workloads.png

[14] How .NET MAUI works, ©2022. *Microsoft technical documentation* [online]. Redmont: Microsoft [cit. 2022-02-23]. Dostupné z: https://docs.microsoft.com/en-gb/dotnet/maui/what-is-maui#how-net-maui-works

[15] Supported platforms, ©2022. *Microsoft technical documentation* [online]. Redmont: Microsoft [cit. 2022-02-23]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/maui/supported-platforms

[16] Co .NET MAUI nabízí, c2022. *Technická dokumentace Microsoftu* [online]. Redmont: Microsoft [cit. 2022-02-23]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/maui/what-is-maui#what-net-maui-provides

[17] Jeden projekt, c2022. *Technická dokumentace Microsoftu* [online]. Redmont: Microsoft [cit. 2022-02-23]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/maui/fundamentals/single-project

[18] Úvod do technologie .NET, © 2022. *Technická dokumentace Microsoftu* [online]. Redmont: Microsoft [cit. 2022-02-23]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/core/introduction

[19] What's new in .NET 6, © 2022. *Microsoft technical documentation* [online]. Redmont: Microsoft [cit. 2022-02-18]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/core/whats-new/dotnet-6

[20] Mobile Device, © 2022. *Techopedia* [online]. Techopedia [cit. 2022-03-10]. Dostupné z: https://www.techopedia.com/definition/23586/mobile-device

[21] What is Operating System? Explain Types of OS, Features and Examples: What is an Operating System?, c2022. *Guru99* [online]. Guru99 [cit. 2022-03-10]. Dostupné z: https://www.guru99.com/operating-system-tutorial.html

[22] How the Android Platform Architecture Works. *Android Developer* [online]. [cit. 2022-03-10]. Dostupné z: https://www.androiddeveloper.co.in/blog/how-android-platform-architecture-works/

[23] IOS, c2022. *Techopedia* [online]. Janalta Interactive [cit. 2022-03-10]. Dostupné z: https://www.techopedia.com/definition/25206/ios

[24] *IOS Architecture* [online], 2017. [cit. 2022-03-10]. Dostupné z: https://aruniphone-application.blogspot.com/2018/08/ios-architecture.html

[25] Mobile Application (Mobile App), ©2022. *Techopedia* [online]. Janalta Interactive [cit. 2022-03-10]. Dostupné z: https://www.techopedia.com/definition/2953/mobile-application-mobile-app

[26] Unified and extended platform, c2022. *.NET Blog* [online]. Microsoft [cit. 2022-03-21]. Dostupné z: https://devblogs.microsoft.com/dotnet/announcing-net-6/

[27] What is .NET?, c2022. *.NET* [online]. Microsoft [cit. 2022-03-21]. Dostupné z: https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet

[28] What is .NET Framework?, c2022. *.NET* [online]. Microsoft [cit. 2022-03-21]. Dostupné z: https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet-framework

[29] What is Xamarin.Forms?, c2022. *Microsoft technical documentation* [online]. Microsoft [cit. 2022-03-21]. Dostupné z: https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin-forms

[30] What is Flutter and Why You Should Learn it in 2020. *FreeCodeCamp* [online]. Free Code Camp [cit. 2022-03-21]. Dostupné z: https://www.freecode-camp.org/news/what-is-flutter-and-why-you-should-learn-it-in-2020/

[31] *Flutter documentation* [online]. [cit. 2022-03-21]. Dostupné z: https://docs.flutter.dev/

[32] *Xamarin documentation* [online], c2022. Microsoft [cit. 2022-03-21]. Dostupné z: https://docs.microsoft.com/en-us/xamarin/

[33] ASP.NET Core Blazor, c2022. *Microsoft technical documentation* [online]. Microsoft [cit. 2022-03-21]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/blazor/?WT.mc_id=dotnet-35129-website&view=aspnetcore-6.0

[34] Blazor: Build client web apps with C#, c2022. *.NET* [online]. Microsoft [cit. 2022-03-21]. Dostupné z: https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor

[35] *Introduction to Ionic* [online]. [cit. 2022-03-21]. Dostupné z: https://ionicframework.com/docs

[36] *Capacitor: Cross-platform Native Runtime for Web Apps* [online]. [cit. 2022-03-21]. Dostupné z: https://capacitorjs.com/docs

[37] Developer Survey: Other frameworks and libraries. *Stack Overflow* [online]. [cit. 2022-03-24]. Dostupné z: https://insights.stackoverflow.com/survey/2021#most-popular-technologies-misc-tech-prof

[38] Web application (Web app), c2006-2022. *TechTarget* [online]. TechTarget [cit. 2022-03-24]. Dostupné z: https://www.techtarget.com/searchsoftwarequality/definition/Web-application-Web-app

[39] ROOMI, Mishal, c2022. 5 Advantages and Disadvantages of Web Application. *HitechWhizz* [online]. HitechWhizz [cit. 2022-03-24]. Dostupné z: https://www.hitechwhizz.com/2021/04/5-advantages-and-disadvantages-drawbacks-benefits-of-web-application.html

[40] ROOMI, Mishal, c2022. 7 Advantages and Disadvantages of Hybrid Apps. *HitechWhizz* [online]. HitechWhizz [cit. 2022-03-24]. Dostupné z: https://www.hitechwhizz.com/2021/04/7-advantages-and-disadvantages-limitations-benefits-of-hybrid-apps.html

[41] KHOROSHULIA, Stanislav. What is React Native?. *Mobidev* [online]. [cit. 2022-03-24]. Dostupné z: https://mobidev.biz/blog/how-react-native-app-development-works

[42] PATERSKA, Patrycja, c2022. What is React Native And When to Use It For Your App?. *EL Passion* [online]. [cit. 2022-03-24]. Dostupné z: https://www.elpassion.com/blog/what-is-react-native-and-when-to-use-it

[43] *React Native: Learn once, write anywhere.* [online], c2022. Meta Platforms [cit. 2022-03-24]. Dostupné z: https://reactnative.dev/

[44] Controls, c2022. *Microsoft technical documentation* [online]. Microsoft [cit. 2022-04-09]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/maui/user-interface/controls/

[45] Random Class, c2022. *Microsoft technical documentation* [online]. Microsoft [cit. 2022-04-25]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/api/system.random?view=net-6.0

[46] .NET MAUI Shell overview, c2022. *Microsoft technical documentation* [online]. Microsoft [cit. 2022-04-25]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/maui/fundamentals/shell/

[47] Asynchronous programming with async and await, c2022. *Microsoft technical documentation* [online]. Microsoft [cit. 2022-04-26]. Dostupné z: https://docs.microsoft.com/sk-sk/dotnet/csharp/programming-guide/concepts/async/

[48] Display pop-ups, c2022. *Microsoft technical documentation* [online]. Microsoft [cit. 2022-04-27]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/maui/user-interface/pop-ups

[49] Frame, c2022. *Microsoft technical documentation* [online]. Microsoft [cit. 2022-05-03]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/maui/user-interface/controls/frame

[50] Behaviors, c2022. *Microsoft technical documentation* [online]. Microsoft [cit. 2022-05-04]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/maui/fundamentals/behaviors

[51] Binding value converters: Binding converter properties, c2022. *Microsoft technical documentation* [online]. Microsoft [cit. 2022-05-04]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/maui/fundamentals/data-binding/converters

[52] Pass arguments, c2022. *Microsoft technical documentation* [online]. Microsoft [cit. 2022-05-04]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/maui/xaml/pass-arguments

[53] Switch, c2022. *Microsoft technical documentation* [online]. Microsoft [cit. 2022-05-04]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/maui/user-interface/controls/switch

[54] XAML namespaces, c2022. *Microsoft technical documentation* [online]. Microsoft [cit. 2022-05-04]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/maui/xaml/namespaces/

[55] Create a .NET MAUI Shell app, c2022. *Microsoft technical documentation* [online]. Microsoft [cit. 2022-05-06]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/maui/fundamentals/shell/create

[56] Data binding overview (WPF .NET), c2022. *Microsoft technical documentation* [online]. Microsoft [cit. 2022-05-07]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/desktop/wpf/data/?view=netdesktop-6.0

[57] Data binding, c2022. *Microsoft technical documentation* [online]. Microsoft [cit. 2022-05-07]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/maui/fundamentals/data-binding/

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API	Aplikačný interface
OS	Operačný systém
VS	Visual Studio
.NET	Framework pre vývoj software
C#	Programovací jazyk
XAML	Extensible Application Markup Language
iOS	Mobilný operačný systém spoločnosti Apple
macOS	Desktopový operačný systém spoločnosti Apple
UI	Používateľské rozhranie
UX	Používateľská skúsenosť
SDK	Súprava nástrojov na vývoj software
ART	Android Runtime
HAL	Abstraktná vrstva hardware
CSS	Cascading Style Sheet
HTML	HyperText Markup Language

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

Nenašli sa žiadne položky zoznamu obrázkov.

SEZNAM PŘÍLOH