Fakulta elektrotechnická
Katedra aplikované elektroniky a telekomunikací

Diplomová práce

Využití mobilních aplikací - vzorová řešení

Autor práce: Bc. Jan Novák
Vedoucí práce: Ing. Kamil Kosturik, Ph.D.

Plzeň 2019

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: **2018/2019**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Jan NOVÁK**
Osobní číslo:	**E18N0044P**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Elektronika a aplikovaná informatika**
Název tématu:	**Využití mobilních aplikací - vzorová řešení**
Zadávající katedra:	**Katedra aplikované elektroniky a telekomunikací**

Zásady pro vypracování:

Připravte vzorové aplikace pro mobilní platformy navázané na skutečná data z projektů katedry KAE.

1. Uvažujte co nejlepší přenositelnost na různé platformy.
2. Zdroje dat uvažujte buď vlastní databáze nebo data uložená v cloudu.
3. Navržená řešení implementujte jako vzorové aplikace, které bude možné dále použít při výuce.

Rozsah grafických prací: **podle doporučení vedoucího**

Rozsah kvalifikační práce: **40 - 60 stran**

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

Seznam odborné literatury:

Xamarin Documentation [online]. [cit. 2018-05-01]. Dostupné z: https://docs.microsoft.com/en-us/xamarin/

Vedoucí diplomové práce: **Ing. Kamil Kosturik, Ph.D.**
Katedra aplikované elektroniky a telekomunikací

Datum zadání diplomové práce: **5. října 2018**

Termín odevzdání diplomové práce: **30. května 2019**

Prof. Ing. Zdeněk Peroutka, Ph.D.
děkan

Západočeská univerzita v Plzni
Fakulta elektrotechnická

L.S.

Doc. Dr. Ing. Vjačeslav Georgiev
vedoucí katedry

V Plzni dne 5. října 2018

Abstrakt

Tato diplomová práce je zaměřena na využití mobilních aplikací a na jejich multiplatformní vývoj. Jsou zde popsány základní technologie ovlivňující využití mobilních zařízení a aplikací. Větší část práce se zaměřuje na multiplatfromní vývoj pomocí framework Visual .NET Xamarin. V rámci této práce byla také vytvořena multiplatfromní aplikace DiplomApp na platformy Android, iOS a Windows. Pro vývoj této vzorové aplikace byl využit Xamarin.

Klíčová slova

mobilní aplikace, Xamarin, Android, iOS, Windows Phone, multiplatformní vývoj, .NET framework, Visual Studio, Xamarin.Forms, SQLite, webové služby

Abstract

Novák, Jan. *Mobile applications - sample solutions* [*Využití mobilních aplikací - vzorová řešení*]. Pilsen, 2019. Master thesis (in Czech). University of West Bohemia. Faculty of Electrical Engineering. Department of Applied Electronics and Telecommunications. Supervisor: Kamil Kosturik

---

Main subject of this Master thesis is the use of mobile applications and their crossplatform development. Ground technologies affecting the use of mobile devices and apps are described. Major part of this thesis is concerned around crossplatform app development using Visual .NET Xamarin framework. Thesis includes sample application named DiplomApp. This app was developed for three major mobile platforms: Android, iOS and Windows using Xamarin.

Keywords

mobile application, app, Xamarin, Android, Windows Phone, multiplatform development, .NET framework, Visual Studio, Xamarin.Forms, SQLite, webservices

Prohlášení

Předkládám tímto k posouzení a obhajobě diplomovou práci, zpracovanou na závěr studia na Fakultě elektrotechnické Západočeské univerzity v Plzni.

Prohlašuji, že jsem svou závěrečnou práci vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 270 trestního zákona č. 40/2009 Sb.

Také prohlašuji, že veškerý software, použitý při řešení této diplomové práce, je legální.

V Plzni dne 30. května 2019

Bc. Jan Novák

.....

Podpis

Poděkování

Tímto bych rád poděkoval vedoucímu diplomové práce Ing. Kamilu Kosturikovi Ph.D. a také Ing. Petru Weissarovi Ph.D. za cenné profesionální rady a připomínky při vypracovávání této práce.

Obsah

Seznam obrázků

Seznam symbolů a zkratek

SoC	System on Chip. Systém na čipu.
CPU	Central Processing Unit. Centrální procesor.
GPU	Graphics Processing Unit. Grafický procesor.
IPU	Interconnect Processing Unit. Propojovací procesor.
RAM	Random Access Memory. Paměť s přímým přístupem určená ke čtení i zápisu.
ROM	Read Only Memory. Paměť uchovávající data i bez napájení, určená pouze ke čtení.
GPIO	General Purpose Input/Output. Vstup/výstup pro všeobecné použití.
VS	Visual Studio.
UWP	Universal Windows Project.
OS	Operating system. Operační systém.
PC	Personal computer. Osobní počítač.
MS	Microsoft.
WP	Windows Phone
VPN	Virtual Private Network. Virtuální soukromá síť.
UI	User Interface. Uživatelské rozhraní.
SDK	Software Development Kit. Sada pro vývoj softwaru.
API	Application Programming Interface. Rozhraní pro programování aplikací.
IDE	Integrated Development Enviroment. Integrované vývojové prostředí.

š

1

Úvod

Pojem mobilní aplikace v současnosti jistě zná většina lidí. Téma mobilních aplikací je velice populární a vzhledem k rozšíření chytrých mobilních zařízení mezi lidmi tato popularita zřejmě jen tak neskončí. Stejně jako všechny jiné programy, mobilní aplikace těží z technologického pokroku a vyvíjejí se stále dál. Jako u osobních počítačů neexistuje jediný operační systém, jediná platforma, která by pod sebe zahrnovala veškerá zařízení a programy, stejně tak není pouze jedna platforma pro mobilní aplikace. Tato práce se zabývá nejen využitím moderních mobilních aplikací, ale také jejich multiplatformním vývojem.

První kapitola se zabývá využitím mobilních aplikací. Samotný jednoduchý výčet možných použití by zcela jistě přesáhl rozsah této práce a stále by nebyl kompletní. Tato práce se na téma využití mobilních aplikací tedy dívá z jiného pohledu, z pohledu dostupných mobilních technologií. Stručným přehledem historie lze zasadit moderní aplikace do kontextu, jakým vývojem si mobilní hardwarové technologie i softwarové aplikace prošly. V této kapitole jsou dále rozebrány základní technologie obsažené v moderních mobilních zařízeních. Ty jsou definující pro současný stav v tomto technologickém odvětví a přímo ovlivňují možná využití těchto zařízení a mobilních aplikací na nich běžících.

Druhá kapitola se věnuje multiplatformnímu vývoji moderní mobilní aplikace. Obsahuje rychlý přehled nejrozšířenějších mobilních platforem. Po seznámení s těmito hlavními platformami je představen mocný nástroj pro multiplatformní vývoj. Zbytek kapitoly se podrobněji věnuje popisu tohoto nástroje, jeho jednotlivým součástím a vlastnostem.

Poslední třetí kapitola je věnována vzorové aplikaci, která byla vytvořena jako součást této práce. Aplikace byla koncipována jako multiplatformní pro tři nejrozšířenější platformy. Byla vyvinuta pomocí multiplatformního vývojového frameworku představeného ve druhé kapitole a obsahuje několik funkcí, na kterých demonstruje tento druh vývoje. Kapitola tuto vzorovou aplikaci podrobně popisuje a věnuje se také vlastnostem a problémům spojeným se zvoleným vývojovým prostředím.

2

Možnosti využití mobilních aplikací

Mobilní aplikace jsou softwarové aplikace navržené a vyvinuté pro chytrá mobilní zařízení. Mezi takové počítáme chytré telefony, tablety a spousty dalších mobilních zařízení. V současnosti se staly mobilní aplikace nedílnou součástí moderní společnosti. Téměř každý člověk vlastní a používá nějaké chytré mobilní zařízení. Aplikace jsou nástroji pro organizaci a usnadnění každodenních povinností jak v osobním, tak v profesním životě. Velice důležitou roli hrají v komunikaci, vzdělání, socializaci a dopravě. Díky připojení k internetu jsou neomezeným zdrojem informací na dosah ruky a v neposlední řadě také slouží k poskytnutí zábavy. Díky svojí univerzálnosti postupně nahradily spoustu zařízení určených pouze k jedné činnosti. Využití mobilních aplikací je přímo závislé na technologickém pokroku zařízení, na kterých pracují. S představením nových technologií se vždy najdou jejich nové aplikační využití a stávající aplikace se také vyvíjejí. Trend vývoje aplikací směřuje k vytváření vysoce specializovaných aplikací a jimi využívaných služeb. [1]

2.1 Historie mobilních aplikací

Historie mobilních aplikací se začala psát s vyvinutím prvního mobilního telefonu obsahujícího mikročipy, které pro uskutečňování hovorů využívaly software. Tento první telefon byl poprvé použit v roce 1973 a obsahoval velmi jednoduchý program. Nepodporoval aplikace tak jak je známe dnes, ale šlo o významný první krok k moderním mobilním zařízením. V následující éře mobilních telefonů znamenaly mobilní aplikace pouze jednoduché kalendáře, kalkulačky apod. Stále se nejednalo o aplikace jak je známe dnes, ale zabudované součásti firmware daného zařízení. [2, 3]

První opravdu mobilní aplikace přišly s kapesními počítači od firmy Psion. Tato zařízení (většinou PDA) obsahovala operační systém EPOC. První 16-bitové verze vytvořené začátkem devadesátých let zahrnovaly aplikace jako textový procesor, databáze a tabulky. Novější verze s 32-bitovým OS obsahovaly také 2MB RAM a umožňovaly uživatelům přidat do zařízení další aplikace. Tyto rozšiřující aplikace byly dostupné skrze

softwarové balíčky, nebo ke stažení online, pokud už měl uživatel připojení k internetu. Systém EPOC byl programován v jazyce OPL (Open Programming Language), umožňoval přidávání uživatelských aplikací a stal se základem pro pozdější populární Symbian OS.[2, 3]

Na rozdíl od ustáleného prostředí tradičních mobilních telefonů otevřela chytrá zařízení s operačními sytémy prostor pro aplikace od vývojářů třetích stran. Mobilní společnosti chtěly plně využít nový a rychle se rozvíjející trh a podporovaly vývoj většího a většího množství aplikací s cílem zatraktivnit mobilní zařízení pro uživatele. Díky tomuto rychlému rozvoji mobilních zařízení a aplikací existují pokročilé a intuitivní zařízení tak, jak je známe dnes.[1]

V červenci 2008 otevřela společnost Apple oficiálně svůj iOS App store, první distribuční službu pro mobilní aplikace. Určila tím standard pro distribuci aplikací ostatním společnostem, díky čemuž dnes najdeme podobnou službu na každé mobilní platformě. V případě dvou nejrozšířenějších operačních systémů (Android a iOS) si dnes mohou uživatelé vybrat z knihovny milionů aplikací, které si mohou jednoduše stáhnout do svého zařízení.

2.2 Technologie

Moderní chytré telefony a tablety lze identifikovat jako malé počítače. Rozdíl mezi stolními počítači a těmito zařízeními je velký, nicméně sdílí spolu také spoustu paralel. Stejně jako desktopy obsahují procesor, paměti, zpracovávají vstupy a generují výstupy a využívají operační systém, na kterém běží funkční programy (aplikace). Každé moderní chytré mobilní zařízení je tvořeno podle architektury SoC. To znamená, že základem těchto zařízení je integrovaný obvod, který zahrnuje veškeré součásti počítače do jediného čipu. Tyto čipy mohou u moderních zařízení obsahovat i více než 30 hardwarových částí - CPU, GPU, IPU, RAM, ROM, GPIO a mnoho dalších. V příloze na obr. A.1 lze nahlédnout do analýzy struktury SoC čipu z blíže nespecifikovaného mobilního zařízení Apple. Každá společnost, která vyvíjí a navrhuje vlastní SoC, drží podrobnosti jejich konstrukce v tajnosti [4]. Z fotografií jejich odhalené struktury lze nicméně odhalit alespoň základní blokovou strukturu.

Většina mobilních zařízení využívá architekturu ARM. Hlavním důvodem pro využití této procesorové architektury je menší spotřeba energie. Procesory ARM pracují s méně instrukcemi (procesory založené na RISC - počítač s redukovaným počtem instrukcí), díky čemuž mohou být tyto procesory fyzicky menší a jednodušší, což se projevuje právě ve spotřebě. Z toho důvodu také mohou být chlazeny pouze pasivně a přenosná zařízení tak neobsahují aktivní chladiče.

Využití mobilních aplikací zcela závisí na dostupných technologiích obsažených v zařízení, na které jsou vyvinuty. V následujících kapitolách jsou popsány klíčové technologie a vlastnosti současných mobilních zařízení, které definují podobu většiny mobilních aplikací.

2.2.1 Spotřeba energie

Důležitým faktorem při využití mobilních aplikací je spotřeba energie. Mobilní zařízení jsou napájeny akumulátory. Kvůli tomu jsou jejich systémy navrženy a optimalizovány pro co nejnižší spotřebu energie. I přes takový návrh je ale spotřeba těchto zařízení stále relativně vysoká a to díky aplikacím běžícím na pozadí a díky principu fungování jejich hardware. I když se postupně objevují nová a inovativní řešení jako například bezdrátové nabíjení a hardware se stále zdokonaluje a snižuje se jeho spotřeba, problém s životností nabitého akumulátoru stále přetrvává. Každý uživatel musí počítat pouze s omezeným časem používání zařízení před jeho vybitím. Jedním z největších faktorů je způsob samotného využívání zařízení. Velké množství aplikací spuštěných a běžících na pozadí, stejně tak jako dlouhá doba rozsvíceného displeje akumulátor velmi zatěžuje a rapidně tak snižuje dobu před jeho vybitím. Cílem vývoje těchto zařízení ale samozřejmě není omezování uživatele a tak se hledají stále nová technická řešení. Akumulátory používané v chytrých zařízeních ale mají svá technologická omezení. Aby se vešly do kompaktních mobilních zařízení jsou omezeny svojí velikostí, díky čemuž mohou pojmout pouze určité množství energie. Akumulátory, které obsahují chytrá mobilní zařízení jsou tzv. sekundární napájecí články, lze je dobíjet (na rozdíl od primárních, které mají náboj po sestavení a dobít je nelze). [5, 6]

Lithium-iontový akumulátor je nejpoužívanější technologií u moderních mobilních telefonů. Je to velmi lehký akumulátor s vysokou hustotou uložené energie, díky čemuž dokáže fungovat na vyšším napětí oproti mnoha akumulátorům používajících jiné technologie (3,7V oproti až 1,2V) a dokáže pojmout velké množství energie při relativně malých rozměrech. Jedna z vlastností této technologie je schopnost rychlého nabíjení, kdy lze akumulátor velice rychle nabít až do 80% kapacity. Zbylá kapacita je poté nabita pomaleji. Akumulátor se sám pomalu vybíjí pokud není používán. Další nevýhodou této technologie je také nestabilita při při proražení pláště nebo při přehřátí, které hrozí pokud je akumulátor zkratován. Poté existuje nebezpečí ohně nebo výbuchu. Moderní zařízení využívající tento typ akumulátoru obsahují komplexní napájecí obvody, které těmto nebezpečným stavům zabraňují, úplně vyloučit je ale nikdy nelze. Příkladem chybného designu zařízení s fatálními následky se stal v roce 2016 mobilní telefon Samsung Galaxy Note 7, kdy docházelo ke vznícení a explozím zařízení, což vedlo až ke stažení modelu z projede [7]. [5]

Velikost použitých akumulátorů se liší od zařízení. Čím výkonnější zařízení, tím má zpravidla větší displej a tím větší mechanické rozměry, které umožňují zahrnutí většího

akumulátoru. Velikosti se obvykle pohybují mezi 2000mAh - 4500mAh. Při běžném užívání zpravidla tato kapacita vydrží k užívání zařízení po jeden den před potřebou opětovného nabití. Při velké energetické zátěži, např. využití satelitní navigace a s tím spojený permanentně rozsvícený displej dokáže plně nabité zařízení vybít už za několik hodin. Všechna zařízení ale zpravidla obsahují také úsporný mód, který dokáže prodloužit funkční čas až na několik dní. V tomto módu se zpravidla omezuje připojení zařízení k sítím, jas a rozlišení displeje a jsou zakázané aplikace běžící na pozadí. Využití těchto zařízení tak záleží na dostupnosti nabíjení. Při vývoji aplikace je třeba tento problém zohlednit a optimalizovat aplikaci pro co nejnižší spotřebu energie.

2.2.2 Internetové připojení

Mobilní připojení k síti je jedním z hlavních důvodů rozšíření a popularity chytrých mobilních zařízení. To, že uživatel vlastní zařízení permanentně připojené k internetu, otevřelo dveře k moderním chytrým systémům. Masivní rozšíření těchto zařízení mezi uživatele zcela změnilo způsob jejich využívání a využívání online dat. Využití aplikací s internetovým připojením je prakticky neomezené. Takové aplikace mohou např. stahovat aktuální dopravní informace pro použití k cestování, uživatelé mohou přes aplikace provádět okamžité platby a spravovat svoje finance kdekoliv a kdykoliv apod. Naprostá většina uživatelů těchto chytrých zařízení využívajících online služby vytváří velký tak na mobilní operátory. Ti byli donuceni k investicím a vytvoření moderních vysokorychlostních datových sítí s velkou kapacitou. Mobilní zařízení mají společnou a definující charakteristiku, pohybují se. Potřeba podpory datových služeb pro pohybující se zařízení se stala základem návrhu mobilních sítí, které musí přesměrovat datový provoz na aktuální polohu uživatele. [8]

Současná mobilní zařízení obsahují hned několik síťových rozhraní. Ke klasickému připojení skrze mobilní síť se přidává také rozhraní WiFi. I přes stále se rozšiřující pokrytí WiFi připojení a jeho využívání uživateli ke stahování velkého množství dat je datová spotřeba klasického mobilního připojení stále dominantní. WiFi připojení uživatelé běžně využívají ve svých domovech či pracovištích, při celodenním využívání svého zařízení se ovšem nevyhnou místům bez WiFi. Majoritu mobilního datového toku tvoří TCP a HTTP služby, z nichž nejrozšířenější jsou prohlížení webových stránek a streamování audio vizuálního obsahu. [8]

Operační systémy chytrých zařízení poskytují funkce související se síťovou mobilitou, které mohou aplikace využít. Tyto funkce se liší podle operačního systému. Zahrnují podporu událostí jako je získání a ztráta připojení k síti, nebo příkazy ke zjištění informace o dostupnosti připojení. Samotné připojení je většinou řízeno službou zvanou Connection Manager (manažer připojení, dále CM). Tato služba rozhoduje, které dostupné připojení je momentálně pro zařízení nejvhodnější, vytvoří přístupové rozhraní a předává síťové

informace aplikacím pro další použití. CM a jeho funkcionalita závisí na dané implementaci v daném OS. Například v konzumaci streamovaných médií se Android a iOS chovají rozdílně při požadování dat ze serveru a mají také rozdílná pravidla pro vyrovnávací paměť těchto dat. I když je vývojářská komunita pro tyto nejrozšířenější mobilní platformy početná, neexistuje oficiální dokumentace pro CM a síťový stack. Vývojářská komunita je zaměřena spíše na aplikační vrstvu, kde je vývojář zaměstnán pouze zjištěním, zda je internetové připojení dostupné, ale už ne jeho optimalizací. [8]

2.2.3 Dotykové rozhraní

Mobilní chytrá zařízení existovala s klasickým displejem určeným pouze k zobrazování a mechanickou klávesnicí či jiným vstupem. Teprve s představením dotykového ovládacího rozhraní, dotykovým displejem, se ale dočkala tato zařízení masivního rozšíření mezi uživatele. Tato technologie umožňuje tvorbu velice jednoduchého a uživatelsky přívětivého ovládacího rozhraní, díky kterému si získaly dotykové zařízení svojí velkou popularitu. V dnešní době tvoří mobilní zařízení s dotykovým rozhraním naprostou většinu. Stále existují zařízení, která obsahují také mechanickou klávesnici (hlavně přístroje BlackBerry), nicméně i u těchto zařízení je tento mechanický vstup doplněn o dotykový displej. První mobilní dotykové zařízení obsahovalo LCD displej s dotykovou maticí, která tvořila 16 malých dotykových ploch. Tímto zařízením byl kapesní počítač Casio PB-1000 v roce 1987. Od té doby došlo k výraznému technologickému pokroku, hlavními dvěma technologiemi moderních dotykových displejů jsou technologie odporová a kapacitní.

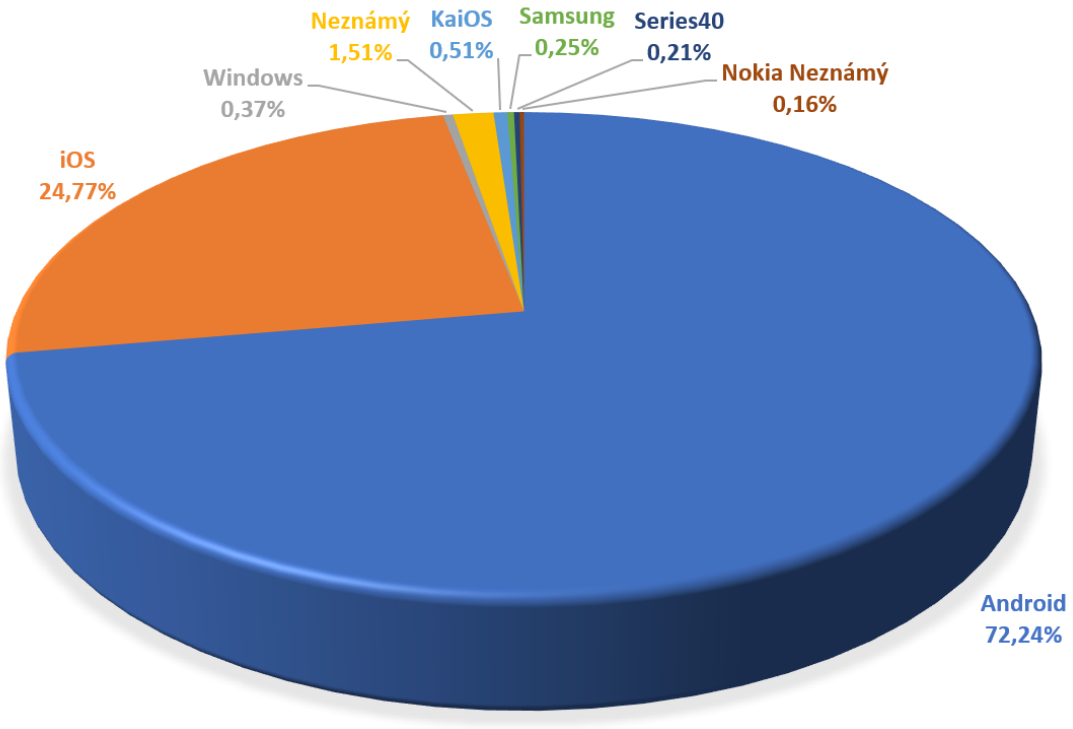
Odporové dotykové displeje byly rozšířené v prvních generacích PDA počítačů a prvních dotykových chytrých telefonech. Stále jsou nejrozšířenější v průmyslových aplikacích, vzhledem k jejich vysoké odolnosti vůči teplotě, vodě a nízké výrobní ceně. K ovládání těchto displejů je často používáno dotykové pero (stylus). Odporové displeje obsahují dvě vrstvy vodivého materiálu, které jsou od sebe odděleny tenkou mezerou. V místě doteku dojde k prohnutí horní vodivé vrstvy a jejímu doteku s vrstvou spodní. Tím dojde k propojení elektrického obvodu. Poloha takto vytvořeného kontaktu je vypočítána z odporu tohoto nově uzavřeného obvodu. Existuje několik druhů odporových dotykových displejů, které jsou rozděleny podle počtu použitých elektrod, mezi kterými se uzavírá hodnocený elektrický obvod. Nejjednodušší je 4-drátový (4-Wire) displej, který obsahuje dvojici elektrod na spodní i horní vrstvě. Další displej obsahuje 5 drátů (5-Wire), 4 elektrody v rozích spodní vrstvy a jedna elektroda uprostřed vrstvy horní. Nejpřesnější je 8-drátová varianta (8-Wire), která je technologicky stejná jako první 4-drátová varianta, ale obsahuje redundantní propojení ke každé elektrodě. Díky této redundanci lze kompenzovat změnu odporu únavou materiálu mezi jedním párem elektrod. Odporové displeje teoreticky umožňují zpracovat více dotyků v jeden okamžik (tzv. multitouch), ale toto zpracování je složité a nevyužívá se, protože by musela být vytvořena matice propojovacích elektrod a jednoduchost displeje by byla ztracena. [9, 10]

Nejrozšířenější technologii dotykových displejů v chytrých telefonech a tabletech tvoří kapacitní displeje. Kapacitní displeje jsou starší technologií než displeje odporové, přesto se staly jejich nástupci ve spotřební elektronice. Na rozdíl od odporových displejů nevyužívají mechanický tlak k zaregistrování doteku, ale přirozenou vodivost lidského těla. Jsou vytvořeny z průhledného vodivého materiálu, který je nanesen na skleněnou vrstvu. Tato skleněná vrstva tvoří vnější materiál, kterého se člověk dotýká. Existují hlavní dva druhy kapacitních dotykových displejů. Prvním druhem je displej využívající tzv. povrchovou kapacitu. U této technologie jsou v rozích displeje umístěny elektrody, na které je přivedeno konstantní napětí. Jakmile dojde k doteku vodivého prstu s jakoukoliv částí displeje, začne mezi prstem a elektrodami téct proud. Díky změně napětí na jednotlivých elektrodách je poté vypočítána poloha prstu. Druhou technologií je tzv promítnutá kapacita. U těchto displejů je vytvořena matice průhledných elektrod v celém displeji. Řádky v jedné ose tvoří řídící rovinu jsou izolovány od řádků matice v druhé ose tvořící snímací rovinu. Na elektrody v řídící rovině je připojen konstantní proud, na elektrodách snímací roviny je poté proud měřen. Při doteku vytvoří vodivý prst propojení v mřížce matice. Tato technologie je jedna z nejcitlivějších a také nejpoužívanějších v moderních zařízeních. Díky vyhodnocování matice elektrod podporuje tato technologie více současných dotyků (multitouch). Výhodou kapacitní technologie oproti technologii odporové je citlivost na velmi lehké doteky (u nejcitlivějších displejů lze detekovat prst i na malou vzdálenost bez výslovného doteku), lepší zobrazovací vlastnosti a mechanická odolnost. Nevýhodou je nutnost dotyku pomocí vodivých předmětů (prsty, nebo speciální stylusy). [9, 10]

3

Multiplatformní vývoj

Existuje velké množství mobilních operačních systémů - platforem, na které lze aplikace vyvíjet. S postupným vývojem přenosných zařízení stále vznikají nové OS a staré, mnohdy dříve velice populární, zanikají. V době psaní této práce většinu trhu s mobilními telefony ovládají hlavně dva operační systémy a to Android a iOS. To lze vidět v grafu na obr. 3.1, data pocházejí ze statistické webové stránky [11].



Obr. 3.1: Tržní podíl operačních systémů chytrých telefonů a tabletů za období 04/2018 - 04/2019. Zdroj dat: [11]

Záměrem vývojářů každé aplikace je zcela jistě její rozšíření mezi maximální množství uživatelů v dané cílové skupině. Jak je patrné z předešlého grafu, vývojem aplikace pro

platformy Android a iOS dokáží vývojáři obsáhnout až 97% trhu s mobilními zařízeními. Vývoj pro více operačních systémů naráz představuje větší výzvu než vývoj pouze na jednu platformu, nicméně se lze vyhnout případným budoucím problémům při portování již vytvořené aplikace. V současnosti je logické vytvořit verze aplikace minimálně pro obě nejrozšířenější platformy.

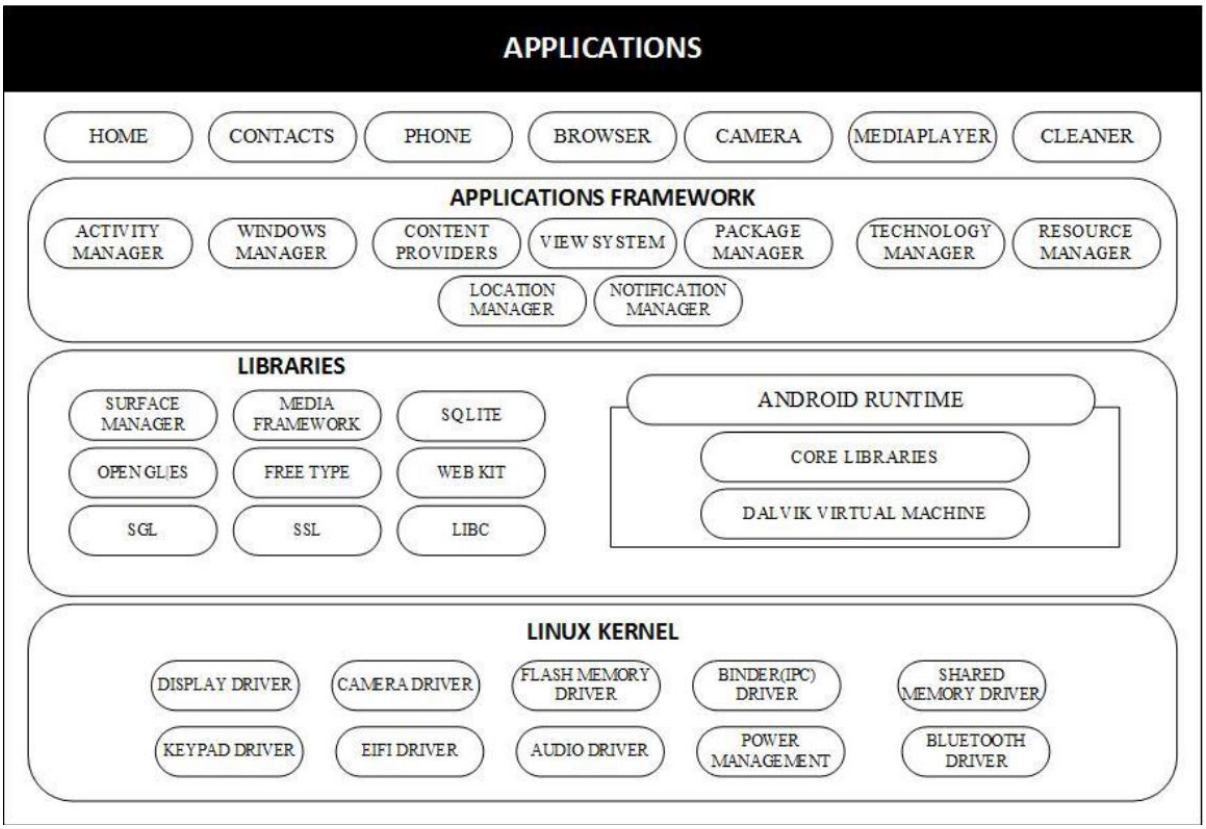
Způsob, usnadňující multiplatformní vývoj, je využití vývojové platformy Visual .Net Xamarin, kterou se zabývá tato práce. Nejdříve ale představení tří mobilních operačních systémů, které byly zahrnuty do vývoje vzorové aplikace k této práci. K nejrozšířenějším systémům Android a iOS byl vybrán ještě OS Windows.

3.1 Android

Jedná se o nejrozšířenější platformu na přenosných zařízeních. Architektura tohoto OS byla poprvé vyvinuta společností Android Inc. (nyní část společnosti Google) v roce 2007, kdy byl spouštěn projekt AOSP (Android Open Source Project). Reakcí na projekt AOSP bylo založení OHA (Open Handset Alliance). Tato aliance je kolekcí softwarových, hardwarových a telekomunikačních firem, které začaly využívat OS Android jako platformu pro svoje mobilní zařízení. Cílem této aliance je technologický pokrok při snížení nákladů a vývojového času, zatím co jsou uživatelům přidávány a vylepšovány funkcionality zařízení. Členy aliance jsou firmy jako Intel, Google, Qualcomm, NVIDIA, HTC nebo T-Mobile. [17]

Android je operační systém založený na Linuxovém jádru (kernel). Na obr. 3.2 je znázorněna struktura tohoto OS. Nejvyšší vrstva je v Android vrstva aplikační, která zahrnuje funkčnosti jako je zasílání SMS zpráv, adresář, kamera a veškeré uživatelem nainstalované aplikace. Vývoj aplikací na OS Android probíhá v programovacím jazyku Java, nebo v C# .NET. K vývoji je potřeba aplikační framework jako základ pro samotnou aplikaci, obsahující základní rutiny a zdroje pro vytvoření responzivního grafického rozhraní aplikace a vnitřní funkcionality.

Android je úzce propojený s kolekcí proprietárního software společnosti Google nazvanou Google Mobile Services. Ta je ve většině případů na zařízení s Android předinstalovaná a obsahuje aplikace pro služby jako jsou Google vyhledávání, mapy, Gmail a hlavně obchod s aplikacemi Google Play. Skrze tuto službu jsou uživatelům jednoduše přístupné veškeré aplikace.



Obr. 3.2: Architektura operačního systému Android. Převzato z: [17]

3.2 iOS

Operační systém společnosti Apple vytvořený pro přenosná zařízení. Na rozdíl od systémů Android a Windows běží iOS pouze na zařízeních vyvinutých touto společností. To umožňuje vysokou úroveň optimalizace, vzhledem k relativně nízkému počtu typů podporovaných zařízení a paralelnímu vývoji hardware a software. Spolu se systémem macOS, který je určen pro počítače a notebooky (také pouze pro zařízení Apple), tak vznikl jedinečný uzavřený ekosystém.

Systém iOS je průkopníkem intuitivního uživatelského rozhraní pro dotykové displeje. Jako první představil koncept moderního UI, který více či méně adoptovaly ostatní mobilní operační systémy. Uživatelské rozhraní je založeno na přímé manipulaci pomocí gest využívajících jeden či více dotykových bodů. Obsahuje také funkce pro usnadnění přístupu pro uživatele s poruchami zraku a sluchu. iOS je postaven na jádru Unix-like XNU, stejně jako systém macOS. XNU je tzv. hybrdní kernel architektura, která spojuje vlastnosti monolitických kernelů a mikrokernelů). [12]

Xamarin umožňuje vývoj aplikací pro iOS. Pokud ale vývojář nepoužívá prostředí Visual Studio for Mac, tedy pokud nepracuje z počítače od firmy Apple osazeným operačním systémem macOS, musí počítat s jistými komplikacemi. Kompilace kódu pro systém iOS totiž vyžaduje Xcode IDE, které obsahuje iOS SDK a běží pouze pod macOS. Při vývoji

aplikace pro iOS se tedy vývojář jisté potřeby počítače se systémem macOS nevyhne. Visual Studio na platformě Windows nabízí možnosti vzdáleného připojení k zařízení Mac a využití tohoto počítače ke kompilaci a spuštění simulátoru zařízení s iOS pro debugging. Zařízení Mac se musí nacházet v lokální síti s počítačem s Windows, na kterém probíhá vývoj v Xamarin. Alternativou je vytvoření virtuálního počítače s macOS na zařízení Windows a ten použít k samotnému vývoji, nebo se na něj připojovat z VS pro Windows. Další alternativou je využití některé z online služeb, které přes internet pronajímají výpočetní čas na zařízeních s macOS, na něž se uživatelé vzdáleně připojují.

3.2.1 Xcode IDE

Vývojové prostředí pro macOS obsahující iOS SDK, které je nutné pro vývoj iOS aplikace. První verze byla představena v roce 2003, poslední stabilní verze (10.2) je zdarma dostupná přes Mac App Store. Xcode nabízí podporu širokému spektru programovacích jazyků (C, C++, Objektové C, Java, Swift a další). Většina současných nativních aplikací je programována v jazyce Java nebo Swift. Xcode umožňuje tvorbu tzv. "tlustých" binárních souborů, umožňující kompilovat zdrojový kód nejen pro systémy iOS běžící na zařízeních s architekturou ARM, ale také pro platformy PowerPC a x86. [13]

3.3 Windows

Windows firmy Microsoft je nejrozšířenější platforma mezi osobními počítači, svojí cestu si ale tento operační systém našel i na přenosná zařízení. Vzhledem k tomu, že základem pro vývoj v Xamarin je Visual .Net, je zahrnutí této platformy logické. V dnešní době jsou ale již mobilní zařízení se systémem Windows na konci svého životního cyklu a tvoří minoritu na trhu. Nejnovější verze systému - Windows 10 vznikla pro sjednocení ekosystému od Microsoftu. Tento systém je navržen jak pro osobní počítače, tak pro přenosná zařízení. Z toho důvodu je systém navržen jak pro klasické ovládací schéma počítače (myš a klávesnice), tak pro dotykové ovládání skrz obrazovku. Vzhledem k momentálnímu stavu trhu a poptávky pro mobilní zařízení se systémem Windows (viz následující kapitola) tvoří nyní většinu dotykově ovládaných zařízení s tímto systémem notebooky s dotykovým displejem, či hybridní zařízení kombinující vlastnosti tabletu a notebooku.

3.3.1 Windows Phone a Windows 10 Mobile

Windows Phone představuje slepou vývojovou větev Windows. Tento systém byl určen pro použití v mobilních telefonech a tabletech, jeho vývoj a podpora již ale byly ukončeny. Pro novější mobilní zařízení s obrazovkou o velikosti do 9 palců byl nahrazen systémem Windows 10 Mobile. Vzhledem k tomu, že WP již není podporován, neobsahuje MS

.NET Standard verzi 2.0. Díky tomu neexistuje zpětná kompatibilita a nově vyvinuté aplikace pod standardem .NET 2.0 již nelze na těchto starších zařízeních spustit. Zánik WP také znamenal konec mobilních telefonů se systémem Windows. Několik zařízení se systémem Windows 10 Mobile sice vzniklo (nebo byl umožněn upgrade ze systému WP), ale vzhledem k podílu těchto zařízení na trhu a ke snižujícímu se zájmu ze strany vývojářů i uživatelů o tyto zařízení bylo již rozhodnuto o ukončení vývoje. Vývoj a podpora Windows 10 Mobile bude ukončena v prosinci 2019.

3.4 Visual .NET Xamarin

Xamarin vznikl v roce 2011 jako společnost s cílem tvorby souboru mobilních projektů. Od svého vzniku využíval projekt Xamarin IDE vývojový jazyk C# ke tvorbě multiplatformních aplikací. V roce 2016 byla společnost vyvíjející Xamarin (Xamarin Inc.) zakoupena společností Microsoft. Následně byl Xamarin SDK převeden na open-source projekt (projekt s volně přístupnými veřejnými zdrojovými soubory) a byl přidán jako nástroj zdarma do IDE MS Visual Studio. [19]

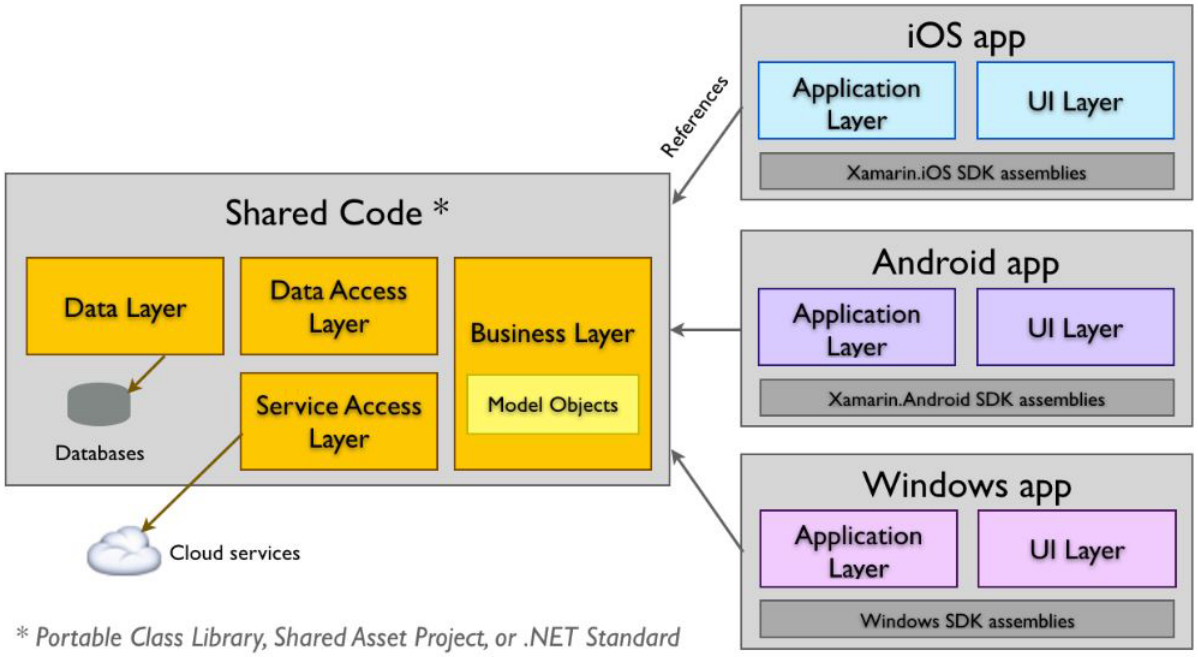
Xamarin framework je platforma, díky které lze vytvořit nativní verze aplikace pro jednotlivé operační systémy (Android, iOS, Windows) v jazyce C#. Odpadá vývoj jednotlivých verzí aplikace v různých jazycích pro různé platformy, veškerá funkcionalita je vytvořena v jednom jazyce a pouze jednou. Tato logika je potom sdílená pro všechny verze. Využitím jazyku C#, knihoven .NET, dostupných API a datových struktur lze dosáhnout až 90% sdíleného kódu mezi jednotlivými platformami. Kompletní propojení s SDK jednotlivých platforem usnadňuje tvorbu robustních a efektivních aplikací a vede k menšímu výskytu chyb. Xamarin umožňuje přímé volání knihoven vytvořených v jazyce C, C++ a Java, což umožňuje využívat širokou škálu kódů vytvořených třetími stranami. Obsahuje také provazující projekty, které umožňují využití nativních knihoven psaných v objektivním C a Java pomocí deklarativní syntaxe. [14, 16]

Xamarin tedy umožňuje vývojářům napsat celou aplikaci v jazyce C# bez ohledu na cílovou platformu a použité knihovny. Tento zdrojový kód je poté kompilován pro každou platformu zvlášť a výsledkem je aplikace s nativním výkonem a vzhledem pro každou jednotlivou platformu. Architektura Xamarin je zobrazena na obr. 3.3. [14, 16]

Popis architektury: [16, 20]

- UI Layer (UI vrstva) - uživatelské rozhraní zahrnuje obrazovku aplikace, ovládání a celkovou prezentaci aplikace uživateli. UI vytvořené v Xamarin vypadají a chovají se jako nativní aplikace pro danou platformu.

- Application Layer (Aplikační vrstva) - specifická pro každou platformu. Propojuje vnitřní logiku aplikace s prezentovaným uživatelským rozhraním.



Obr. 3.3: Architektura Xamarin. Převzato z: [20]

- Business Layer (Funkční logika) - součást sdíleného kódu. Obsahuje implementaci hlavní funkcionality aplikace. Využívá modelové objekty (Model Objects).
- Data Access Layer (Vrstva datových přístupů) - abstraktní vrstva mezi funkční a datovou vrstvou. Zajišťuje přístup funkcionality k datům.
- Service Access Layer (Vrstva přístupu ke službám) - zajišťuje přístupy k externím službám, jako jsou webové služby REST, JSON, WCF. Stará se o připojení a poskytuje rozhraní pro využití těchto služeb v aplikaci.
- Data Layer (Datová vrstva) - obsahuje data (např. databáze SQLite).

Vývoj v Xamarin lze rozdělit na dva druhy, podle principu psaní zdrojového kódu. Při zakládání nového projektu v Xamarin dostáváme na výběr z těchto dvou druhů: .NET Standard, nebo Shared Project (sdílený projekt).

- **Sdílený projekt** - zdrojový kód je v tomto typu projektu kompletně sdílený. Platformně specifické části kódu jsou oddělené direktivou *# if*. Kompiler potom zpracovává pouze relevantní části kódu podle cílové platformy kompilace. Bloky platformně závislého kódu jsou tedy ve stejném souboru se sdílenou funkcionalitou, což má své výhody i nevýhody. Výhodou může být jednodušší debugging, nevýhodou pak složitější a méně čitelnější zdrojový soubor. Ukázka zdrojového kódu psaného ve sdíleném projektu:

```
1  public void PlatformSpecificMethod()
2  {
3    #if __MOBILE__
4    // Specifický kód pro Android nebo iOS
5    #endif
6
7    #if WINDOWS
8    // Specifický kód pro Windows
9    #else
10
11   #if __ANDROID__
12   // Specifický kód pro Android
13   #else
14
15   #if __IOS__
16   // Specifický kód pro iOS
17   #else
18 }
```

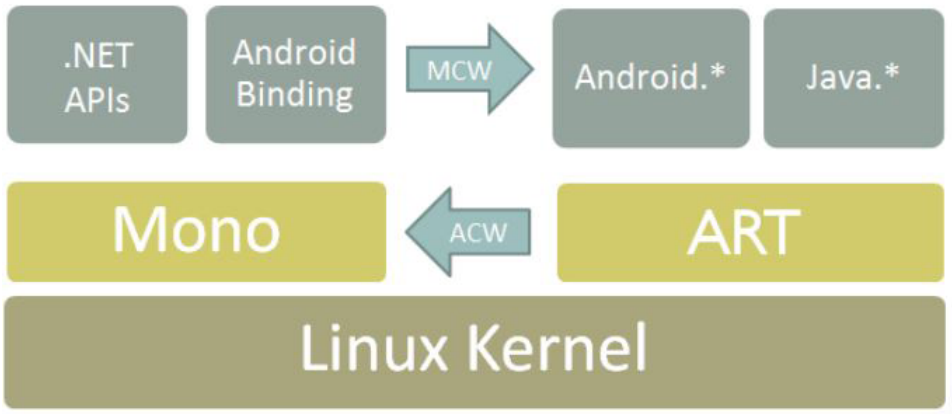
- **.NET Standard** - toto řešení využívá .NET framework se všemi jeho dostupnými API. Obsahuje hlavní projekt aplikace se sdíleným kódem a poté specifické projekty pro každou platformu. Sdílená funkcionalita je implementována v hlavním projektu a skrze implementovaná rozhraní přistupuje k platfromně specifickým částem kódu. Sdílená funkcionalita tak nerozlišuje mezi jednotlivými platformami a interaguje se všemi stejně skrze tato definovaná rozhraní. Při kompilaci použije kompilátor projekt dané cílové platformy jako zdroj kódu pro druhou stranu implementovaných rozhraní. Tento přístup je používanější a také doporučený. Vzhledem k rozdělení platformně specifického kódu do jednotlivých projektů velmi usnadňuje orientaci a čitelnost zdrojového kódu. Umožňuje také poměrně jednoduché dodatečné přidání platformy do již rozpracovaného projektu.

3.4.1 Xamarin.Android

Xamarin.Android je projekt pro android v Xamarin řešení při použití .NET Standard. Obsahuje velké množství API, které usnadňují vývoj a poskytují nativní výkon. Umožňuje tvorbu a využití stejných prvků a funkcionality jako nativní vývoj pro Android v jazyce Java. Vše ale plně podporované pod .NET a v jazyce C# . Pokud vývojář nepoužívá Xamarin.Forms ke tvorbě sdíleného UI, může v tomto projektu vytvořit nativní uživatelské rozhraní pro android využitím XML, nebo přímo ve zdrojovém kódu pomocí C# . [16]

Architekturu Xamarin.Android lze vidět na obr. 3.4. Aplikace Xamarin.Android běží v prostředí Mono, které je implementováno v C# . ART (Android Run Time) virtuální zařízení a prostředí Mono běží paralelně nad vrstvou jádra - Linux Kernel. Díky tomu mají ART a Mono přístupy k rozdílným druhům API jádra pro uživatelský kód. Lze tak např. využít .NET knihovny System.IO, System.Net atp. pro přístup k systémovým funkcím linuxového jádra. Většina systémových funkcionalit (audio, grafika, připojení…) není v Android nativním aplikacím přímo přístupná, ale je k nim přistupováno skrze Android Runtime Java API, které jsou implementovány v namespace Java.* nebo Android.*. ACW

(Android Callable Wrappers) a MCW (Managed Callable Wrappers) jsou přemostění umožňující přístupy mezi kódy Android Runtime a spravovaným uživatelským kódem aplikace. [16, 21]



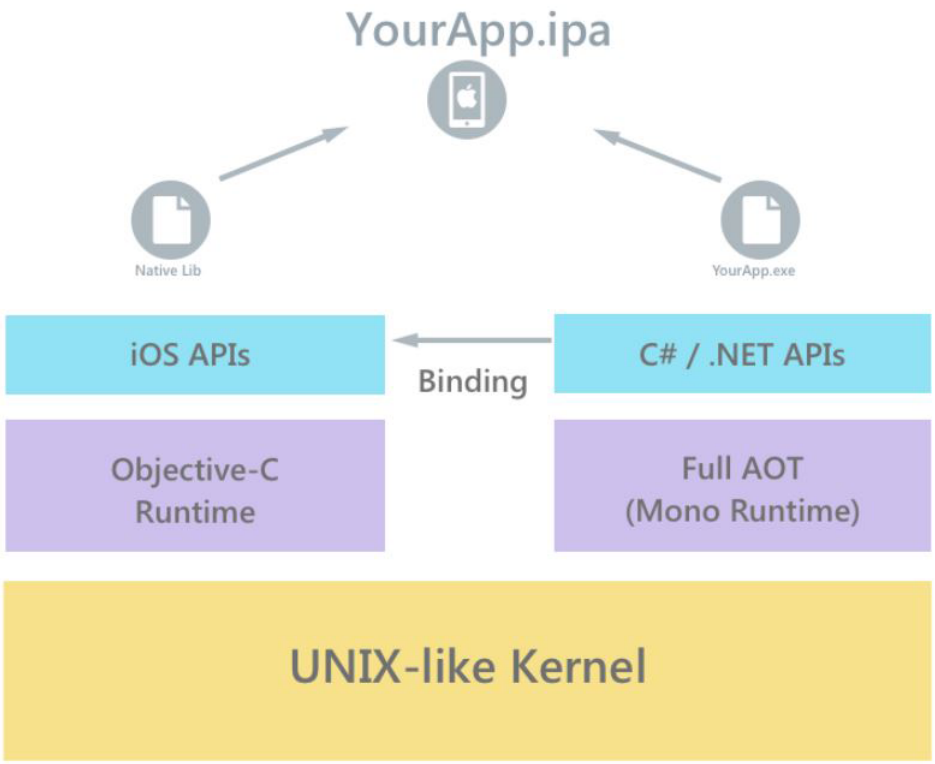
Obr. 3.4: Architektura Xamarin.Android. Převzato z: [21]

Xamarin.Android v IDE Visual Studio dokáže komunikovat se simulátorem zařízení s běžícím systémem Android. Tento simulátor sice není přímou součástí Xamarin, ale lze si jej bezplatně stáhnout. Podpora simulátoru výrazně zjednodušuje vývojový proces a debugging v případě, že vývojář nemá k dispozici žádné fyzické zařízení s tímto systémem. Pokud vývojář vlastní zařízení se systémem android, lze ho připojit k počítači, na kterém probíhá vývoj. Visual Studio poté umožňuje sestavení a debugging aplikace přímo na tomto fyzickém zařízení. Připojením a nastavením zařízení s Android (nebo simulátoru) pro vývoj v Xamarin se blíže věnuje kapitola 4.1.1.

3.4.2 Xamarin.iOS

Xamarin.iOS aplikace také běží v prostředí Mono. Využívá ale AOT kompilaci (Ahead Of Time - kompilace "dopředu") pro kompilaci kódu v C# do sestavovacího jazyka ARM. Mono běží paralelně s runtime v objektovém C. Obě tyto prostředí běží paralelně nad UNIX jádrem. Architekturu Xamarin.iOS si lze prohlédnout na obr. 3.5. Jak je z tohoto obrázku zřejmé, mezi architekturami Xamarin.iOS a Xamarin.Android lze nalézt spoustu paralel.

Velkým rozdílem je využití AOT kompilace. Při kompilování aplikace Xamarin pro jakoukoliv platformu dochází ke zkompilování zdrojového kódu v C# do MSIL (Microsoft Intermediate Language). Pokud je spuštěná aplikace Xamarin.Android nebo Xamarin.iOS v simulátoru, .NET Common Language Runtime (CLR) kompiluje daný kód v MSIL použitím přístupu JIT (Just In Time - kompilace "přesně na čas"). V runtime je takto získán



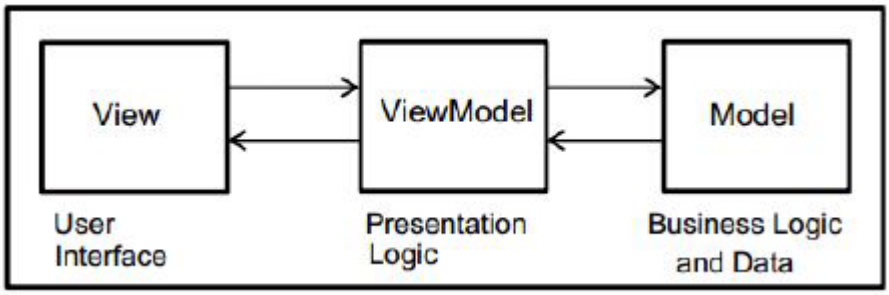
Obr. 3.5: Architektura Xamarin.iOS. Převzato z: [22]

nativní kód běžící na specifické platformě. Z bezpečnostních důvodů není v iOS povolené vykonávání dynamicky generovaného kódu. Toto omezení se nevztahuje na simulátor zařízení iOS, proto je v daném případě také použita kompilace JIT. Aby bylo toto bezpečnostní nařízení splněno při běhu aplikace na reálném zařízení, využívá tedy Xamarin.iOS AOT kompilaci zdrojového kódu před samotným runtime aplikace. Díky tomu získáváme nativní iOS binární soubor, který je spouštěn na procesorech ARM v zařízeních Apple. [21]

3.4.3 MvvmCross (design pattern)

MvvmCross je návrhový vzorec využívaný při tvorbě aplikací v Xamarin. Tento vzorec vychází ze vzorce Mvvm obsahující části: Model - View - ViewModel. Je upravený pro multiplatformní využití zdrojového kódu postaveného jako Mvvm. Strukturu tohoto vzorce lze vidět na obrázku obr. 3.6. MvvmCross rozděluje logiku uloženou v obrazovkce aplikace do tří objektů: View, ViewModel a Model (samotná obrazovka, její model a model logiky). Objekt View poté tvoří uživatelské rozhraní a ViewModel obsahuje jádro, třídy obsahující funkční logiku sdílenou mezi všemi platformami. Model obsahuje další funkční logiku a datový obsah přístupný z více ViewModel objektů. [16]

Struktura MvvmCross využívá datové provázání mezi View a ViewModel, což velice zjednodušuje vývoj pro všechny platformy. Například k provázání ListView (seznam) a pří-



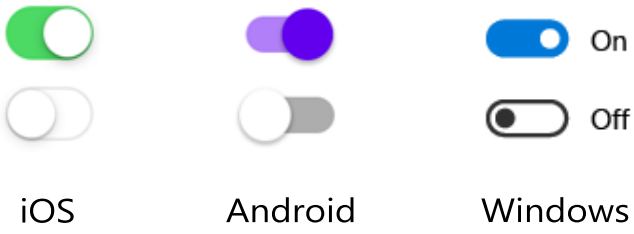
Obr. 3.6: Vývojová struktura MvvmCross. Převzato z: [16]

slušných dat je v Android potřeba vytvoření adaptéru. Při vývoji s použitím MvvmCross se toto provázání jednoduše vytvoří v XAML souboru při tvorbě UI (objekt View). Toto provázání poté funguje pro všechny platformy využívající daný XAML zdrojový soubor k sestavení uživatelského rozhraní. Datové provázání (Data Binding) tvoří obousměrnou komunikaci mezi View a ViewModel. [16]

3.4.4 Xamarin.Forms

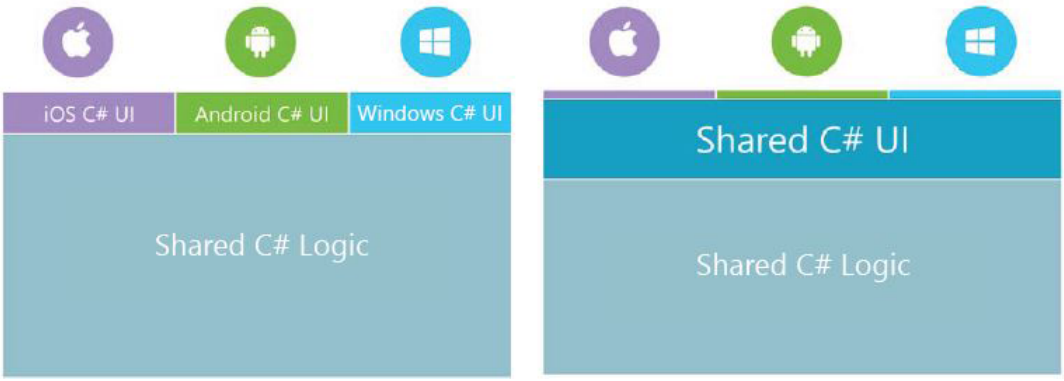
Při tvorbě multiplatformní aplikace s pomocí Xamarin Forms lze dosáhnout opravdu vysokého procenta sdíleného kódu. Xamarin Forms umožňují vytvoření nativního uživatelského rozhraní pro každou platformu z jednoho zdrojového kódu v aplikacích, které vyžadují méně funkcí specifických pro jednotlivé platformy. Takto vytvořené UI může sdílet až 100% kódu. Pro návrh uživatelského rozhraní v Xamarin Forms se používá značkovací jazyk XAML nebo jazyk C# . Preferovaný způsob je většinou tvorba UI v XAML a využití zdrojového kódu v C# pouze pro vytvoření silně podmíněných UI prvků, nebo pro specifické případy, kdy by bylo vytvoření daných prvků pomocí XAML velmi složité až nepraktické. Uživatelské rozhraní obsahuje propojení s funkčním zdrojovým kódem aplikace skrze datové provázání - viz předchozí kapitola. Po kompilaci v runtime využívá rozhraní tvořené pomocí Xamarin Forms nativní UI prvky dané specifické platformy. Pokud například chceme v aplikaci použít přepínač, využijeme element <*switch*> v návrhu UI v XAML. Tento element je pro každou platformu v runtime nahrazen korespondujícím nativním prvkem - třídou UISwitch pro iOS, ToggleSwitch pro UWP a Switch pro Android. Vizuální reprezentaci těchto nativních UI prvků lze vidět na obr. 3.7. Díky tomuto systému lze tedy velice jednoduše navrhnout nativní uživatelské rozhraní, které je přitom jednotné napříč všemi platformami. [16, 18]

Tvorba uživatelského rozhraní v Xamarin Forms je volitelná, vývojáři mají možnost vytvořit UI pro každou platformu zvlášť v jednotlivých projektech (Xamarin.Android atd.). Rozdíl ve struktuře aplikace při použití a nepoužití Xamarin Forms ukazuje obr. 3.8. V případě tvorby UI pro každou platformu zvlášť musí vývojáři použít pro návrh jazyk C# . Funkční logika je potom sdílená stejně, jako v případě využití Xamarin Forms. Při tomto



Obr. 3.7: Vizuální podoba přepínače - nativního prvku uživatelského rozhraní v jednotlivých platformách.

vývoji lze očekávat dosažení kolem 70% sdíleného kódu mezi platformami. Tento přístup je výhodný pro dosažení relativně snadného a rychlého vývoje při zachování možnosti tvorby specifických UI pro jednotlivé platformy. [18]



Obr. 3.8: Struktura aplikace vyvinuté v Xamarin při tvorbě UI pro jednotlivé platformy zvlášť a při použití Xamarin Forms (vpravo). Převzato z: [18]

3.4.5 NuGet balíčky

Každý vývojář pracující s C# .NET jistě zná NuGet balíčky. NuGet je správce balíčků rozšíření, který je zabudovaný do IDE Visual Studio. Díky tomuto správci je velice jednoduché získat nebo také vytvořit knihovny a rozšíření pro právě vyvíjenou aplikaci. Vývojář tak může jednoduše vyhledat a použít funkcionalitu tvořenou oficiálními zdroji (Xamarin tým, .NET tým) nebo jinými řadovými uživateli. Pokud vývojář vytvoří vlastní knihovnu, např. implementací jisté funkcionality optimalizované pro určitý případ, může tuto knihovnu skrze NuGet nabídnout k veřejnému využití. Díky tomuto systému lze najít NuGet balíčky pro většinu běžně využívaných funkcionalit a systémů ve velkém množství iterací a rozdílných implementací. Vzhledem k obsažení Xamarin přímo v IDE Visual Studio a oficiální podpoře je množství NuGet balíčků určených pro mobilní multiplatformní vývoj značné.

3.4.6 Xamarin Live Player

Xamarin Live Player je nástroj pro živý náhled vyvíjeného kódu v Xamarin Forms. Jedná se o aplikaci, která se nainstaluje do mobilního zařízení a po spárování s Visual Studiem potom spouští vyvíjený kód živě na daném zařízení. Live player byl vytvořen jako rychlý způsob vizualizace tvořeného kódu pro vývojáře. Tento nástroj má ovšem velké limitace a spoustu funkcionality lze na zařízení vyzkoušet až po kompilaci. Původně byl zamýšlen pro Android i pro iOS, verze pro iOS ale nakonec nebyla vytvořena. S novou verzí Visual Studio 2019 již není Xamarin Live Player podporován a tím končí jeho oficiální podpora od Microsoftu. [15]

4

Vzorová aplikace

Součástí práce je také vzorová aplikace, vytvořená pomocí Xamarin. Aplikace byla vyvinuta jako ukázka práce s Xamarin při tvorbě multiplatformního software a to na tři mobilní operační systémy. Těmito systémy jsou Android, iOS a Windows. Následující kapitola se zabývá popisem samotné aplikace a jejím vývojem.

4.1 Vývojové prostředí

Pro tvorbu vzorové aplikace bylo využito vývojové prostředí Visual Studio 2017 Community edition (build 15.9.11). Tato verze Visual Studia je zdarma dostupná pro studenty, vývojáře jednotlivce či open source projekty. Pro vývoj byl použit osobní počítač s operačním systémem Windows 10 Home (verze 1803 build 17134.765). Dále byl k vývoji využit počítač od firmy Apple (viz 4.1.2), software Hamachi (také 4.1.2) a software Android Studio (kapitola 4.1.1). Pro ladění aplikace na specifických platformách bylo využito fyzické zařízení LGE LG-H930 (Android 8.0 - API 26), simulátor iPhone 8Plus (iOS 11.2) a samotný počítač (již zmíněný Windows 10 Home).

4.1.1 Ladění pro Android

Pro debugging Android aplikace nabízí IDE Visual Studio vývojáři dvě možnosti. Připojení fyzického zařízení s operačním systémem Android, nebo využití emulátoru daného zařízení. Emulátor zařízení Android je volně dostupný jako součást vývojového prostředí Android Studio.

Pro využití fyzického zařízení Android k účelu vývoje aplikace je potřeba zařízení i počítač správně nastavit. Nastavení zařízení lze shrnout do třech kroků:

1 Povolení debuggingu v zařízení - pro vývoj a ladění lze použít každé zařízení obsahující Android, je ale nutné nejdříve debugging povolit. Tato možnost je v každém zařízení

defaultně zakázaná. Povolení debuggingu je součástí Vývojářského nastavení, které je v Androidu verze 4.2 a vyšší skryté pro běžného uživatele. Toto nastavení se odemyká pomocí položky "číslo sestavení" zařízení, které lze obvykle nalézt ve struktuře **Nastavení - Info o zařízení - Softwarové informace**. Jakmile je tato položka nalezena, stačí na ní 7x poklepat, čímž se odemkne a zviditelní Vývojářské nastavení. V něm je poté zaškrtávácí položka **Ladění USB** povolující debugging skrze připojený USB kabel.

2 Instalace USB ovladačů - pro správnou funkci ladění aplikace na zařízení připojené k počítači pomocí USB je potřeba, aby systém Windows obsahoval správné a aktuální ovladače pro dané zařízení. Tyto ovladače jsou obvykle k dispozici na internetových stránkách výrobce daného zařízení. V případě nedostupnosti ovladačů od výrobce zařízení je možnost využití USB ovladačů obsažených v Android SDK, které je volně šířené a je také obsažené v IDE Android Studio.

3 Propojení zařízení s počítačem - po fyzickém propojení počítače a zařízení pomocí kabelu USB by toto zařízení mělo být úspěšně rozpoznáno systémem Windows. Pokud jsou nainstalovány funkční ovladače a je povolené USB ladění, objeví se toto zařízení v Visual Studio IDE v seznamu zařízení využitelných k ladění.

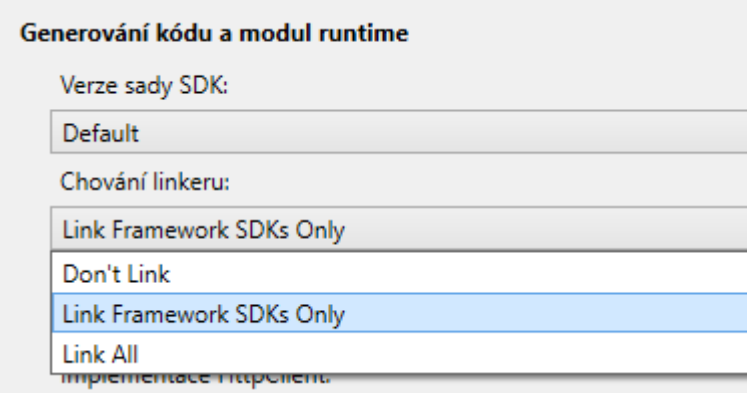
Emulátor zařízení s Android dokáže běžet ve velkém množství konfigurací. Lze v něm takto emulovat rozličná zařízení s měnícími se parametry jako je například rozlišení. Vzhledem k velkému množství zařízení s tímto systémem je tato schopnost emulátoru měnit svoje parametry velice užitečná pro testování napříč celým spektrem zařízení. Pro využití tohoto emulátoru je třeba instalace vývojového prostředí Android Studio. Toto prostředí je volně šířené vývojáři systému Android a obsahuje v sobě Android SDK a zmíněný emulátor zařízení. Po nainstalování je vhodné IDE spustit a otevřít v něm položku **Android Virtual Device Manager**. V okně manažeru zařízení se systémem Android můžeme prohlížet vlastnosti jednotlivých zařízení (konfigurací emulátoru) a také jednotlivé zařízení přidávat nebo mazat. Pokud proběhla instalace správně a lze emulátor spustit z IDE Android Studio, po spuštění IDE Visual Studio budou dostupné konfigurace emulátoru obsažené v seznamu zařízení pro ladění Xamarin aplikace.

4.1.2 Vzdálený přístup k macOS

Jak bylo zmíněno v sekci 3.2, pro vývoj projektu na platformu iOS v operačním systému Windows je potřeba vzdálený přístup k počítači Mac. Pro účely vývoje iOS verze diplomové aplikace byl zapůjčen MacBook Pro (verze Early 2015) obsahující macOS Sierra (verze 10.12.6). Tato již starší verze systému macOS je zdrojem jistých konfliktů ve vývojovém toolchainu, viz konfigurace počítače Mac dále.

Zapůjčený notebook nemohl být provozován na lokální síti s počítačem, na kterém probíhal vývoj, musel tedy být využit software pro vytvoření VPN. Tím byl LogMeIn Hamachi. Jedná se o SW pro vytvoření šifrované VPN, který je zdarma při jistých omezeních (počet zařízení v síti a počet vytvořených sítí). Pro účely této diplomové práce, tj. propojení dvou zařízení, je tento uživatelsky jednoduchý SW zcela vhodný. Více informací o Hamachi a také samotný program lze získat na webové adrese [23].

Po úspěšném propojení zařízení v lokální síti je třeba správně nakonfigurovat Mac i Visual Studio. Základem je povolení vzdáleného připojení na zařízení Mac, díky čemuž se lze na tento počítač připojit. Dále musí zařízení obsahovat instalovaný software Visual Studio for Mac včetně rozšíření Xamarin a také Xcode IDE. Verze Visual Studio se nemusí shodovat, musí se ale shodovat verze Xamarin.iOS SDK. Vzhledem ke starší verzi operačního systému macOS nebylo možné nainstalovat nejnovější verzi Xcode, ale byla nainstalována nejvyšší verze podporovaná na daném macOS (Xcode 9.2 build 9C40b). Tato starší verze Xcode způsobuje chybu při pokusu o kompilaci. Použitá verze Xamarin.iOS požaduje iOS 12.2 SDK (obsažený v Xcode 10.2). Vzhledem k nemožnosti aktualizace daného SDK lze tuto chybu obejít pomocí nastavení linkeru pro DiplomApp.iOS projekt. Ve vlastnostech projektu lze v menu iOS - sestavení nastavit chování linkeru k propojení SDK frameworku (obr. 4.1). Toto nastavení umožňuje kompilaci pomocí staršího iOS SDK, vývojář si ovšem musí dát pozor na využití API, které jsou implementovány pouze v novějších verzích.



Obr. 4.1: Nastavení linkeru iOS projektu pro provázání iOS SDK verzí mezi počítačem, na kterém probíhá vývoj, a počítačem Mac, na kterém běží vzdálená kompilace a simulátor. Vhodné pro případ, kdy verze Xamarin.iOS vyžaduje rozdílnou verzi iOS SDK od verze instalované na počítači Mac.

4.2 DiplomApp

Vzorová aplikace DiplomApp byla vytvořena jako Xamarin aplikace využívající řešení .NET Stadard. V tomto typu řešení existuje sdílený projekt pro jádro aplikace vedle

specifických projektů pro každou dílčí platformu, viz 3.4. Zdrojový kód aplikace využívá návrhový vzorec MvvmCross a nativní uživatelské rozhraní bylo vytvořeno pomocí Xamarin.Forms. Aplikace zahrnuje několik funkcionalit tvořící průřez možnostmi frameworku Xamarin.

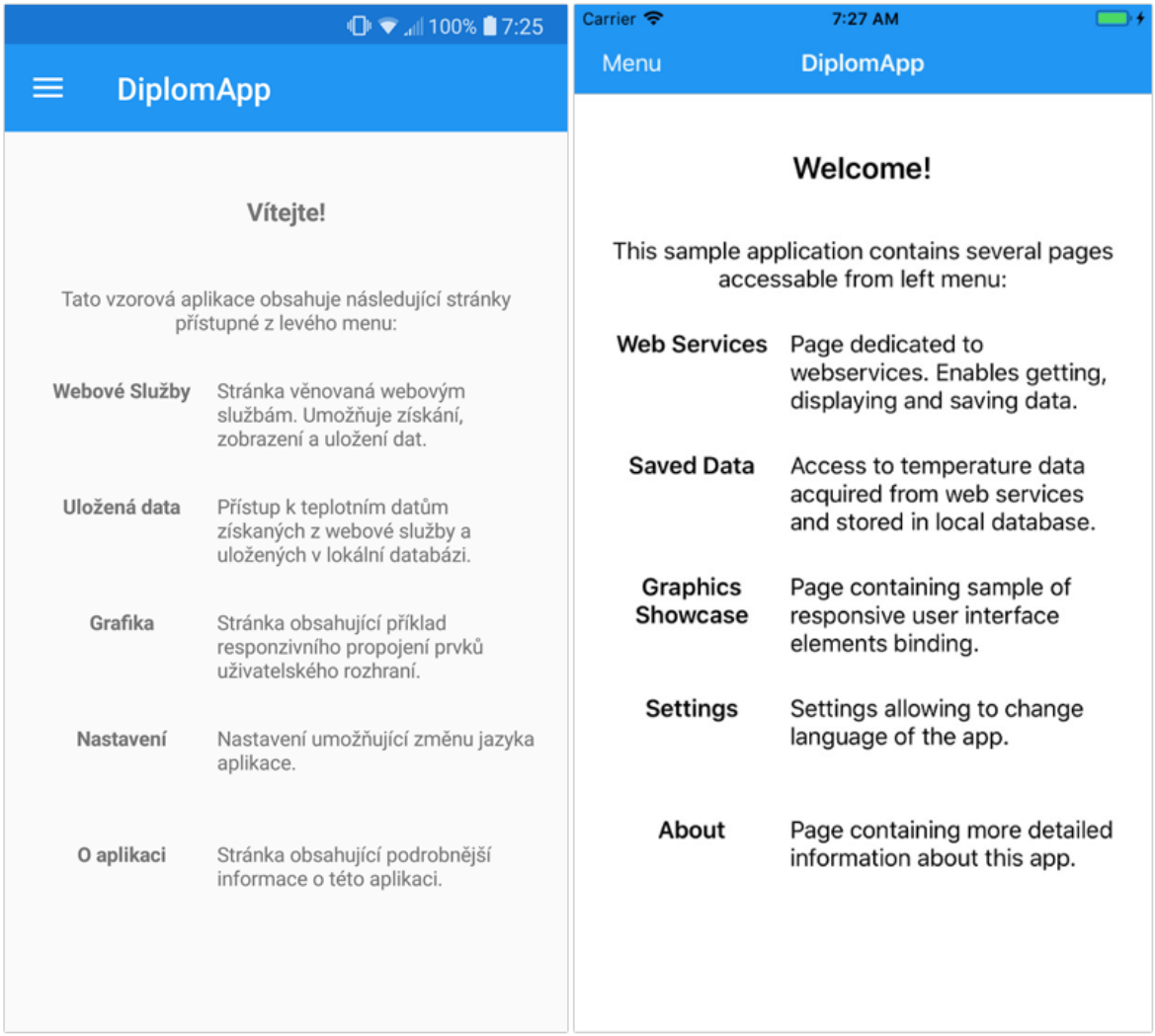
V řešení byly zahrnuty následující balíčky NuGet:

- NETStandard.Library v2.0.3, Autor: Microsoft
- System.ServiceModel.Duplex v4.4.0, Autor: Microsoft
- System.ServiceModel.Http v4.4.0, Autor: Microsoft
- System.ServiceModel.NetTcp v4.4.0, Autor: Microsoft
- System.ServiceModel.Security v4.4.0, Autor: Microsoft
- Xamarin.Forms v3.4.0.1039999, Autor: Microsoft
- Xamarin.Android.Support.Design v27.0.2.1, Autor: Xamarin Inc.
- Xamarin.Android.Support.v4 v27.0.2.1, Autor: Xamarin Inc.
- Xamarin.Android.Support.v7.AppCompat v27.0.2.1, Autor: Xamarin Inc.
- Xamarin.Android.Support.v7.CardView v27.0.2.1, Autor: Xamarin Inc.
- Xamarin.Android.Support.v7.MediaRouter v27.0.2.1, Autor: Xamarin Inc.
- Microsoft.NETCore.UniversalWindowsPlatform v6.1.12, Autor: Microsoft
- SQLiteNetExtensions v2.1.0, Autor: TwinCoders

Aplikace je dělená do několika stránek, které obsahují jednotlivé funkce. Úvodní stránka poskytuje rozcestník s krátkým popisem a lze si ji prohlédnout na (obr. 4.2).

4.2.1 Provázání zdrojového kódu dle MvvmCross

Prezentované uživatelské rozhraní tvoří stránka (např. *ConnectivityPage*), která patří mezi objekty *ContentPage* - stránky zobrazující jednotlivá *View*. Uživatelské rozhraní je vytvořeno v jazyce XAML v souboru **ConnectivityPage.xaml** a je provázáno s odpovídajícím modelem *ConnectivityViewModel* pomocí vlastnosti *BindingContext* (vm je namespace obsahující jednotlivé modely):



Obr. 4.2: DiplomApp, úvodní stránka aplikace poskytující přehled implementovaných funkcí, platformy Android a iOS (vpravo).

```
<ContentPage.BindingContext>
  <vm:ConnectivityViewModel />
</ContentPage.BindingContext>
```

Takto nastavený kontext určuje, že všechna provázání vytvořená direktivou *Binding* v tomto XAML souboru odkazují do *ConnectivityViewModel*. Pomocí *Binding* tak lze přistupovat k libovolné veřejné funkci nebo proměnné definované v propojeném modelu. V následující části kódu si lze prohlédnout provázání řetězce textu do UI popisku a také provázání funkce obsluhující stisk tlačítka a povolení jeho funkce:

```
<Button Text="{Binding Resources[btnSend]}}"
        Command="{Binding BtnNameCommand}"
        IsEnabled="{Binding BtnNameEnabled}"/>
<Label Text="{Binding LblName}" />
```

Stisknutí tlačítka lze obsluhovat dvěma způsoby. Prvním způsobem je přiřazení funkce obsluhující událost kliknutí. Tato funkce se přiřazuje pomocí vlastnosti *Clicked="Btn_ Clicked"*. Funkce *Btn_ Clicked* je poté vytvořena ve zdrojovém kódu *View* a provázání s *ViewModel* lze provést uvnitř této funkce. Druhým způsobem je přímé provázání z XAML do *ViewModel*. Tento způsob je použit v DiplomApp a lze jej vidět v ukázkách kódu. Spojení je provedeno přes vytvořený příkaz *Command* který je navázán na tlačítko. V modelu je k tomuto příkazu přiřazená funkce obsluhující stisk tlačítka (*BtnNameClicked*). Implementace v *ConnectivityViewModel* je poté:

```
public ICommand BtnNameCommand { get; private set; }

public ConnectivityViewModel()
{
  BtnNameCommand = new Command(BtnName_Clicked);
}

public async void BtnName_Clicked()
{
 ...
}
```

Pro správnou funkci navázání proměnných z *ViewModel* do zdrojového kódu v XAML je třeba v modelu vytvořit přístupové rozhraní obsahující *get;* a *set;*. Důležitou součástí tohoto rozhraní je použití *OnPropertyChanged* při nastavení proměnné. Díky tomu je při každém nastavení dané proměnné aktualizováno UI a to potom odpovídá vždy aktuální hodnotě. Implementaci propojení zdrojového kódu *ConnectivityViewModel* odpovídajícího předchozí ukázce XAML:

```
private bool _BtnNameEnabled = true;
private string _LblName;

public string LblName
{
  get { return _LblName; }
```

```
7    set
8    {
9      _LblName = value;
10     OnPropertyChanged(nameof(LblName));
11   }
12 }
13
14 public bool BtnNameEnabled
15 {
16   get { return _BtnNameEnabled; }
17   set
18   {
19     _BtnNameEnabled = value;
20     OnPropertyChanged(nameof(BtnNameEnabled));
21   }
22 }
```

Provázání UI skrze *Binding* do daného rozhraní obsahující *get;* i *set;* může fungovat obousměrně. Aktivní prvek uživatelského rozhraní tak může nejen reagovat na obsah provázané proměnné, ale také tuto proměnou měnit. Příkladem takového provázání je využití přepínače. Přepínač čte hodnotu provázané boolean proměnné a podle ní mění svojí vizuální polohu (zapnuto/vypnuto). Je tedy vždy v podobě odpovídající aktuální hodnotě této proměnné. Pokud uživatel přepínač stiskne, díky obousměrnému provázání změní hodnotu dané proměnné (a následně tedy i vizuální polohu přepínače). Ukázka implementace obousměrného provázání přepínače:

```
1  //XAML View
2  <Switch IsToggled="{Binding SwitchToggled, Mode=TwoWay}">
3
4  //C# ViewModel
5  private bool _SwitchToggled;
6
7  public bool SwitchToggled
8  {
9    get { return _SwitchToggled; }
10   set
11   {
12     _SwitchToggled = value;
13     OnPropertyChanged(nameof(SwitchToggled));
14   }
15 }
```

Implementace funkční logiky je tedy v daném *ViewModel*, vlastní *View* - v tomto případě tedy třída stránky *ConnectivityPage* obsahuje pouze inicializační funkci, která vytvoří UI z příslušného XAML zdrojového kódu. Tato struktura umožňuje vytvoření libovolného počtu a/nebo instancí stránek - *View* provázaných s jediným *ViewModel*, využívající tedy stejnou funkční logiku. *ConnectivityViewModel* využívá vytvořené modely *TempItem* a *TempList* pro ukládání dat z webové služby. Je tak demonstrována celá struktura MvvmCross.

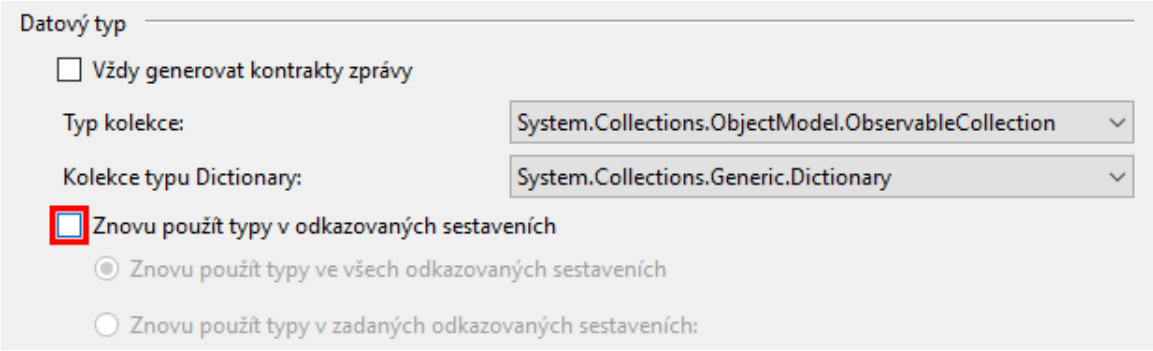
4.2.2 Webové služby (webservices)

Webové služby jsou často používanou funkcionalitou v mobilních aplikacích. Jejich implementace do aplikace dobře ilustruje strukturu Xamarin se sdíleným funkčním kódem

přistupujícím skrze vytvořené přístupové rozhraní do platformně specifických částí kódu. DiplomApp využívá webovou službu vytvořenou pro výukové účely na serveru katedry. Tato služba nabízí připojenému klientovi k využití dvě funkce: získání názvu dané služby a získání naměřených teplotních dat.

Implementaci webových služeb do Xamarin aplikace velice ulehčuje IDE Visual Studio. Umožňuje přidání těchto služeb přímo do jednotlivých projektů uvnitř řešení, přičemž automaticky vygeneruje programové rozhraní pro konzumování těchto služeb. Samotné Visual Studio tedy připraví funkce pro připojení a využití dané webové služby. Vývojáři poté již stačí takto vytvořené API použít. Pro projekty Xamarin.Android a Xamarin.iOS lze přímo přidat webový odkaz na dané služby. Odkaz se přidá pravým kliknutím na projekt a výběrem položky "přidat - webový odkaz". V nově otevřeném okně se poté vyplní http adresa webové služby. Visual Studio na dané adrese vyhledá použitelné služby, které nabídne k použití. Vývojáři následně stačí webovou službu pojmenovat a potvrdit přidání do projektu.

Xamarin.UWP projekt v Xamarin řešení přidání webových služeb přes přímý odkaz neumožňuje. Webová služba se tedy do Windows projektu přidává jako neutrální "přidat odkaz na službu". Pro správné připojení webové služby je třeba v následném okně, kde se vyplňuje adresa služby, otevřít upřesňující nastavení a odškrtnout volbu "znovu použít typy v odkazovaných sestaveních" viz obr. 4.3. Tato možnost je defaultně zaškrtnutá a způsobuje chybu kompilátoru.



Obr. 4.3: Upřesňující nastavení pro přidání odkazu na službu

Takto tedy byly přidány webové služby pod názvem *WebServiceASE*. Visual Studio generuje pro webový odkaz a připojenou službu rozdílné obsluhující funkce. Připojením přes webový odkaz jsou vygenerovány synchronně spouštěné funkce, zatímco pro odkaz na službu jsou vytvořeny asynchronně volané funkce běžící ve vlastním vlákně.

Jako rozhraní mezi sdíleným kódem, který využívá webovou službu, a jednotlivými platformně specifickými implementacemi je vytvořena služba (service) *ISoapService* v

hlavním sdíleném projektu. Tato služba poskytuje propojení ke oběma funkcím použité webové služby:

```
Task<string> GetServiceFullNameAsync();
Task<DataTable> GetLastTempAsync(int lines);
```

Jak je patrné, jedná se o asynchronní funkce. V projektu pro každou platformu byla vytvořena jejich vlastní implementace ve třídě, která byla na všech platformách shodně pojmenována jako *ServiceASE*. Implementace *ServiceASE* je pro Android a iOS prakticky shodná. Vzhledem k tomu, že pro službu připojenou přes webový odkaz nejsou generovány asynchronní obslužné funkce, je spouštění těchto funkcí asynchronně vyřešeno zde. Implementace v projektu Android:

```
[assembly: Dependency(typeof(DiplomApp.Droid.ServiceASE))]
namespace DiplomApp.Droid
{
    public class ServiceASE : ISoapService
    {
        public async Task<DataTable> GetLastTempAsync(int lines)
        {
            WebServiceASE.data_ws service = new WebServiceASE.data_ws();
            return await Task.Run(() => service.GetLastTemp(lines));
        }

        public async Task<string> GetServiceFullNameAsync()
        {
            WebServiceASE.data_ws service = new WebServiceASE.data_ws();
            return await Task.Run(() => service.ServiceFullName());
        }
    }
}
```

Připojení přes odkaz na službu poskytuje asynchronní obslužné funkce, implementace *ServiceASE* pro projekt UWP tedy nepotřebuje řešit spouštění funkcí jako nový *Task*:

```
[assembly: Dependency(typeof(DiplomApp.UWP.ServiceASE))]
namespace DiplomApp.UWP
{
    class ServiceASE : ISoapService
    {
        public async Task<DataTable> GetLastTempAsync(int lines)
        {
            WebServiceASE.data_wsSoapClient service = new WebServiceASE.data_wsSoapClient();
            return await service.GetLastTempAsync(lines);
        }

        public async Task<string> GetServiceFullNameAsync()
        {
            WebServiceASE.data_wsSoapClient service = new WebServiceASE.data_wsSoapClient();
            return await service.ServiceFullNameAsync();
        }
    }
}
```

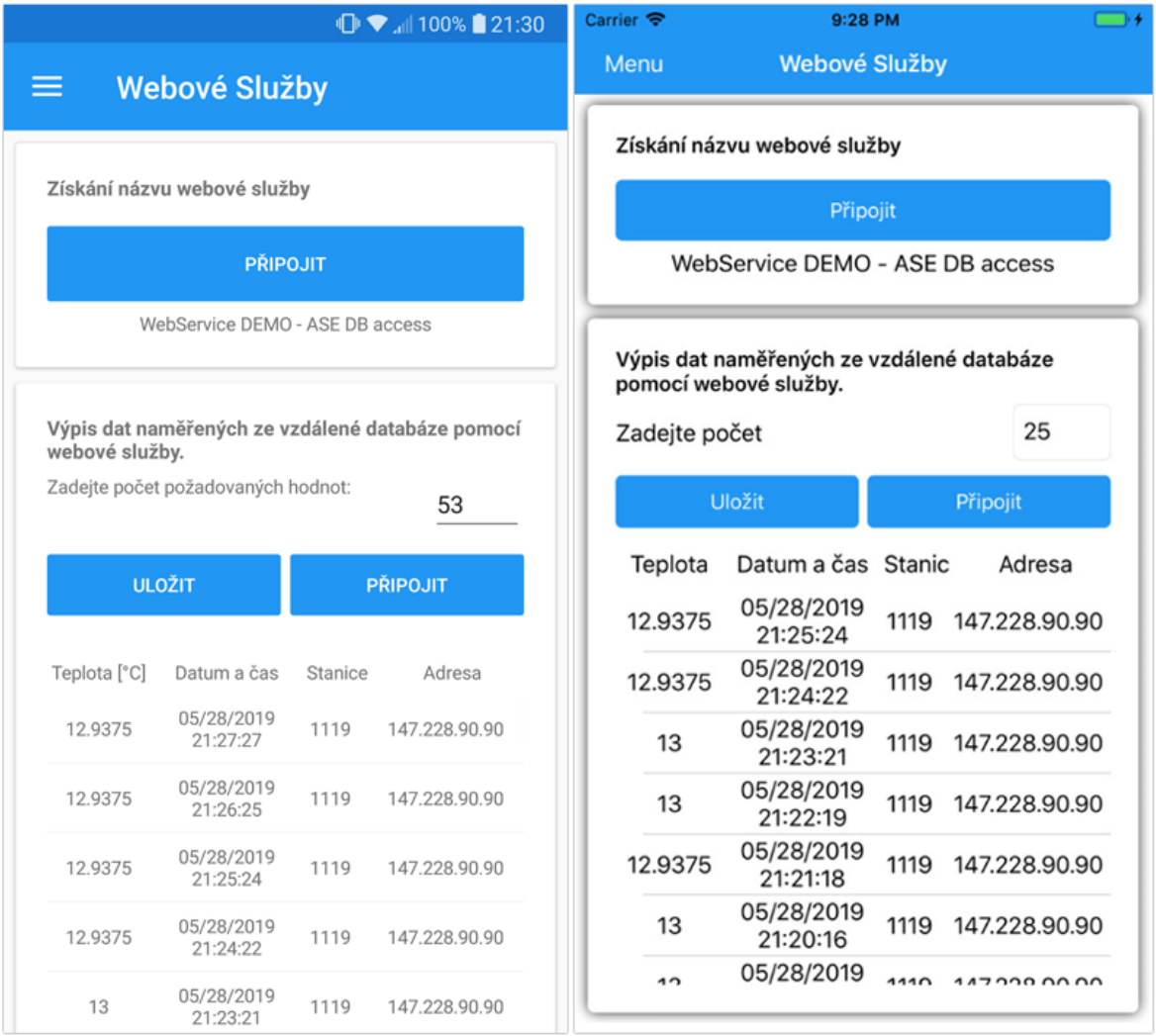
Ve sdíleném zdrojovém kódu je následně využíváno rozhraní *ISoapService*, tento funkční kód nemá žádné informace o platformě, na které je spouštěn. Rozhraní je použito pomocí *DependencyService* a implementace vypadá v DiplomApp následovně:

```
1  LblName = await CallWebServiceName();
2
3  private async Task<string> CallWebServiceName()
4  {
5      var service = DependencyService.Get<ISoapService>();
6      string str;
7
8      try
9      {
10        str = await service.GetServiceFullNameAsync();
11     }
12     catch (Exception)
13     {
14        str = Resources["errConnFailed"] + " (" + DateTime.Now.ToLongTimeString() + ")";
15     }
16
17     return str;
18 }
```

Voláním této webové služby tedy získáváme řetězec obsahující název služby. Pokud se vyskytne chyba připojení či jiná chyba, je místo názvu služby navrácen chybový řetězec obsahující čas chyby. Obdobně je řešena druhá funkce *GetLastTempAsync* (viz dále). Spouštění webových služeb a jiných (potenciálně) časově náročných funkcí by mělo být vždy řešeno jako asynchronní metody. Tím dojde k oddělení této logiky do jiného vlákna než ve kterém běží uživatelské rozhraní. To je důležité pro zaručení funkce UI i během provádění výpočtů na pozadí aplikace. Pokud by k tomuto oddělení nedošlo, uživatelské rozhraní by během provádění operací na pozadí nereagovalo na vstupy a aplikace by se jevila jako zamrznutá.

Stránkou aplikace DiplomApp obsluhující webové služby je již zmíněná *ConnectivityPage*. Je graficky rozdělena na dvě karty, jedna pro každou webovou službu. Stránku si lze prohlédnout na obr. 4.4. Horní karta obsahuje popisek informující o dané funkci a tlačítko "Připojit". Funkce tlačítka implementovaná ve *ViewModel* je velmi jednoduchá a je popsána v předchozím odstavci. Zavolá webovou službu a vypíše vrácený řetězec do popisku pod tlačítkem. Připojení k webové službě je ošetřeno pomocí try - catch a tak pokud není daná webová služba dostupná, nebo se vyskytne jiná chyba, je funkcí vrácen chybový řetězec obsahující také aktuální čas.

Druhá karta připadá k webové službě vracející požadovaný počet měření teploty ze stanice na katedře. Kromě textu s popisem této služby obsahuje tato karta také vstup pro zadání počtu požadovaných měření. Tento vstup má nastavenou numerickou klávesnici (požadujeme číslo), nicméně např. kvůli zařízení Windows (nebo jinému s mechanickou klávesnicí) je tento vstup ještě ošetřen pomocí *Tryparse*. Pokud je zachycen nečíselný nebo nulový vstup, volá se webová služba s požadavkem na jediný záznam. Dále UI obsahuje tlačítko pro připojení a pro uložení získaných dat do databáze. Pod tlačítky se nachází popisek, který zobrazuje chybu při neúspěšném připojení (stejně jako v předchozím funkci), který je při úspěšném načtení dat prázdný. Pod tímto popiskem začíná zobrazení získaných dat. Dokud stránka neobsahuje žádná data, je toto zobrazení skryté. Jakmile ale



Obr. 4.4: DiplomApp, obrazovka webových služeb, platformy Android a iOS (vpravo).

webová služba úspěšně vrátí požadovaná data, je zobrazena tabulka popisků a pod ní seznam obsahující jednotlivá měření.

Webová služba vrací naměřená data ve formátu *DataTable*. Pro další zpracování je tato tabulka převedena na seznam vytvořených modelů *TempItem*. Tento model obsahuje položky korespondující s jednou řádkou získané tabulky - s jedním měřením. Také obsahuje svoje Id a Id nadřazené tabulky - tyto dvě položky jsou nutné pro ukládání do databáze (viz 4.2.5). Jak tento model vypadá si lze prohlédnout zde:

```
1  public class TempItem
2  {
3      [PrimaryKey, AutoIncrement]
4      public int Id { get; set; }
5
6      public DateTime Time { get; set; }
7      public double Temperature { get; set; }
8      public int Station { get; set; }
9      public string Address { get; set; }
10
11     [ForeignKey(typeof(TempList))]
12     public int TempListId { get; set; }
13 }
```

O převod tabulky *DataTable* na seznam objektů *TempItem* se stará jednoduchá funkce obsahující cyklus:

```
1  public List<TempItem> TempTableToList(DataTable dt)
2  {
3      List<TempItem> tempItems = new List<TempItem>();
4
5      for (int i = 0; i < TempTable.Rows.Count; i++)
6      {
7          var item = new TempItem();
8          item.Id = i;
9          item.Temperature = Convert.ToDouble(dt.Rows[i]["teplota"]);
10         item.Time = Convert.ToDateTime(dt.Rows[i]["cas"]);
11         item.Station = Convert.ToInt32(dt.Rows[i]["stanice"]);
12         item.Address = Convert.ToString(dt.Rows[i]["poznamka"]);
13         tempItems.Add(item);
14     }
15     return tempItems;
16 }
```

Získaný seznam objektů *TempItem* je provázán se seznamem *ListView* v návrhu UI. Tento vizuální seznam poté zobrazuje provázaný seznam pomocí navržené datové buňky. V XAML se navrhne vizuální podoba jedné buňky. V tomto případě buňka obsahuje tabulku popisků, do kterých je navázán obsah jednotlivých vlastností objektu *TempItem*. *ListView* samotný poté vytvoří a naplní takto navrženou buňku pro každý prvek provázaného seznamu. Návrh v XAML lze vidět zde:

```
1  <ListView ItemsSource="{Binding Temp}"
2            FlexLayout.AlignSelf="Auto">
3      <ListView.ItemTemplate>
4          <DataTemplate>
```

```
5             <ViewCell>
6                 <Grid>
7                     <Grid.ColumnDefinitions>
8                         <ColumnDefinition Width="23*" />
9                         <ColumnDefinition Width="30*" />
10                        <ColumnDefinition Width="12*" />
11                        <ColumnDefinition Width="35*" />
12                    </Grid.ColumnDefinitions>
13                    <Label Grid.Column="0"
14                            Text="{Binding Temperature}"
15                            HorizontalTextAlignment="Center"
16                            VerticalTextAlignment="Center"/>
17                    <Label Grid.Column="1"
18                            Text="{Binding Time}"
19                            HorizontalTextAlignment="Center"
20                            VerticalTextAlignment="Center"/>
21                    <Label Grid.Column="2"
22                            Text="{Binding Station}"
23                            HorizontalTextAlignment="Center"
24                            VerticalTextAlignment="Center"/>
25                    <Label Grid.Column="3"
26                            Text="{Binding Address}"
27                            HorizontalTextAlignment="Center"
28                            VerticalTextAlignment="Center"/>
29                </Grid>
30            </ViewCell>
31        </DataTemplate>
32    </ListView.ItemTemplate>
33 </ListView>
```

Za povšimnutí stojí parametr *FlexLayout*, který umožňuje natažení daného *ListView* podle počtu zobrazovaných položek. Vzhledem k tomu, že je toto *ListView* zobrazeno do omezeného prostoru na stránce, je tento seznam scrollovací. *ColumnDefinition* nastavuje relativní šířku sloupců tabulky. Zadaná hodnota následovaná hvězdičkou je vyjádření šířky sloupce v procentech šířky tabulky.

4.2.3 Responzivní lokalizace

Pro lokalizaci aplikace využívají systémovou třídu *CultureInfo*. Přes tuto třídu lze získat nebo nastavit kulturu aplikace, což zahrnuje jazyk uživatelského rozhraní a kulturně specifické formátování např. času a datumu. Změnou kultury tak lze např upravit aplikaci na arabický jazyk, který je formátovaný zprava doleva, nebo změnit tvar datumu z dd/mm/yyyy na mm/dd/yyyy. Pro práci s lokalizací existují NuGet balíčky tvořené týmem Xamarin Microsoftu. Tyto knihovny ale neumožňují responzivní lokalizaci - změnu jazyka aplikace v reálném čase. Pro lokalizaci této aplikace nebyly využity. K dosažení změny jazyka při runtime aplikace je třeba využít dynamicky provázaných zdrojů, v této aplikaci byl využit přístup popsaný v [24]. Lokalizaci aplikace lze rozdělit na tři části.

První částí jsou zdrojové soubory obsahující tabulku řetězců použitých v UI včetně jejich překladů do podporovaných jazyků (resources). Každý podporovaný jazyk aplikace má svůj zdrojový soubor s příponou .resx. Platí zde pravidlo pro pojmenování, kdy se všechny tyto zdrojové soubory jmenují stejně kromě zkratky příslušného jazyka oddělené tečkou na konci názvu. Pro dva podporované jazyky v této aplikaci tak existují dva

zdrojové soubory - **AppResource.cs.resx** (pro češtinu) a **AppResource.en.resx** (pro angličtinu). Řešení také může obsahovat zdrojový soubor bez jazykové zkratky, který se poté chová jako základní volba (v tomto případě tedy **AppResource.resx**). Přidáním zdrojového souboru .resx generuje Visual Studio automaticky metodu Designer. Jedná se o metodu typu *ResourceManager*, která umožňuje dynamický přístup ke zdrojům určeným pro specifickou kulturu. Tvoří rozhraní, skrze které lze najít zdroje pro daný jazyk.

Druhou částí je implementace logiky, která se stará o změnu jazyka. Pro tuto funkcionalitu byla vytvořena třída *LocalizedResources*. Obsahuje jednoduché přístupové rozhraní, skrze které UI získává potřebné řetězce ze zdrojových souborů přes *ResourceManager*. Provázání je tvořeno pomocí indexovaných vlastností (*Indexed Properties*). Samotné rozhraní vypadá následovně:

```
1 public string this[string key]
2 {
3     get
4     {
5         return ResourceManager.GetString(key, CurrentCultureInfo);
6     }
7 }
```

ResourceManager vrací string *key* ze zdrojového souboru odpovídajícího kultuře *CurrentCultureInfo*. *LocalisedResources* dále obsahuje metodu *OnCultureChanged*. Tato metoda využívá rozhraní *INotifyPropertyChanged* pro dynamickou aktualizaci nastavené jazykové kultury aplikace. Tato metoda je volaná při změně jazyku aplikace pomocí *MessagingCenter*. Pro toto použití byla tvořena nová zpráva *CultureChangedMessage*, která obsahuje nově zvolenou jazykovou kulturu. Implementace metody:

```
1 private void OnCultureChanged(object s, CultureChangedMessage ccm)
2 {
3    CurrentCultureInfo = ccm.NewCultureInfo;
4    PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("Item"));
5 }
```

Nyní ještě zbývá vytvoření instance *LocalisedResources* v základním modelu *BaseViewModel*. Přes tuto instanci získají jednotlivé *ViewModel* a *View* přístup k rozhraní *Resources*.

Třetí částí je samotné použití jazykových zdrojů při tvorbě uživatelského rozhraní. Použití vytvořeného ve třídě *LocalizedResources* poté vypadá v XAML takto:

```
1  <Label Text="{Binding Resources[Welcome]}">
```

Řetězec v závorkách [ ] je předán do rozhraní jako *key* sloužící pro vyhledání daného textu. Text v popisku (Label) tedy bude obsah řetězce *Welcome* pro daný jazyk určený pomocí *CurrentCultureInfo*. Obdobně lze využít jazykové zdroje v C# :

```
// Použití v C# kódu View
Label.Text = (BindingContext as BaseViewModel).Resources["Welcome"];

// Použití v C# kódu ViewModel
Label = Resources["Welcome"];
```

Mezi provázáním těchto zdrojů v XAML a v C# je ale zásadní rozdíl. Při vykreslování textu do uživatelského rozhraní přes XAML je zaručená dynamická změna zdrojových jazykových souborů během runtime, změna jazyka se projeví v textu okamžitě. Text popisku přiřazený dle kódu v XAML výše tedy bude vždy v aktuálně nastaveném jazyce aplikace. Při použití přístupu do jazykových zdrojů pomocí kódu C# se ale použije daný jazyk v momentě průběhu programu tímto přiřazením. Pokud tedy použiji přiřazení textu do popisku dle výše uvedeného C# kódu pouze v inicializaci stránky, bude i při změně jazyku aplikace v tomto popisku stále původní řetězec. Změna jazyku se projeví až při opětovné inicializaci, nebo při novém přiřazení textu do tohoto popisku. To není problém např. u vyskakovacích hlášek (varování atp.), které jsou vždy vytvořeny v runtime při potřebě využití, je to ale problém při tvorbě perzistivních prvků uživatelského rozhraní. Ty by tedy měly být všechny vytvořeny v návrhu UI v XAML.

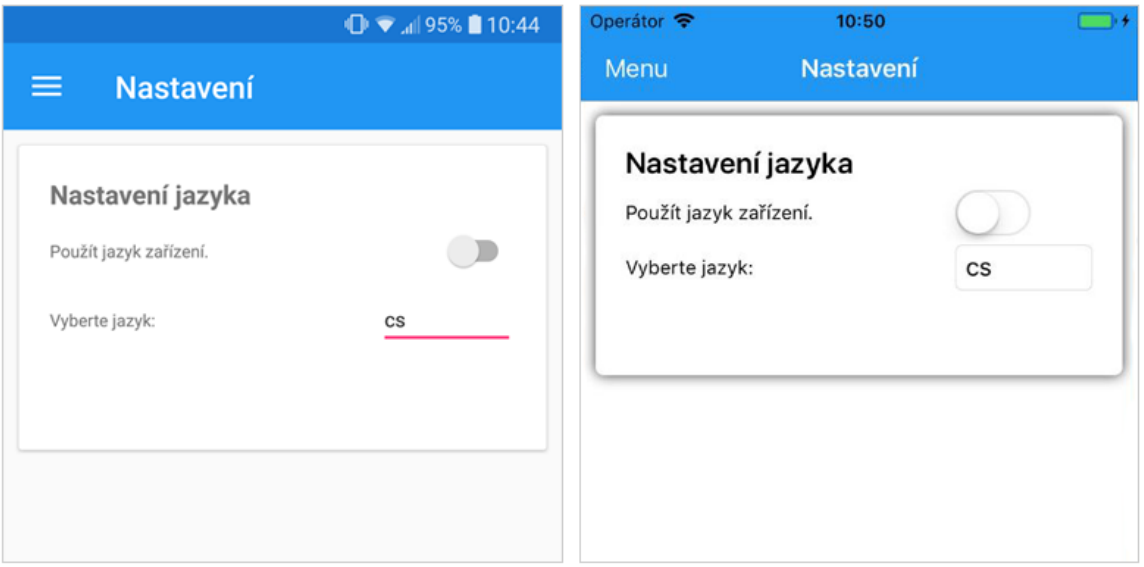
Jsou ale případy, kdy je využití přiřazení textu v C# potřeba pro permanentnější prvky UI. Vysouvací menu je vytvořeno při inicializaci aplikace a využívá C# pro vyplnění jednotlivými položkami. Text položek je přiřazen v kódu při naplnění menu, tedy pouze při inicializaci. Možnost přepsání tohoto menu pro využití XAML existuje, ale momentální implementace v C# je velice výhodná pro rozšiřování tohoto menu a umožňuje jednoduché úpravy během runtime. Aby bylo menu vždy v aktuálním jazyce aplikace, je tedy třeba zavolat inicializaci tohoto menu při změně jazyka. K tomu je také využito *MessagingCenter*, stejně jako pro aktualizaci zvoleného jazyka. Při změně jazyka odešleme zprávu, která způsobí nové naplnění menu položkami, které jsou díky tomu vytvořeny s textem v aktuálním jazyce. Příjem zprávy a nové naplnění menu ve zdrojovém kódu aplikace potom vypadá následovně:

```
MessagingCenter.Subscribe<SettingsViewModel>(this, "LanguageChanged", (sender) => {
PopulateMenu();
ListViewMenu.ItemsSource = menuItems;
});

public void PopulateMenu()
{
  if(menuItems != null) menuItems.Clear();
  menuItems = new List<HomeMenuItem>
  {
    new HomeMenuItem {Id = MenuItemType.Welcome,
                      Title = (BindingContext as BaseViewModel).Resources["pgWelcome"]},
    *
    *
    *
   };
}
```

Jako poslední část už zbývá jen vytvoření stránky nastavení, kde si uživatel bude moct jazyk vybrat. Pro tuto vzorovou aplikaci jsou podporované pouze dva jazyky, angličtina

a čeština. Uživatel si může v rozbalovacím menu mezi těmito jazyky vybrat, nebo si může vybrat zaškrtávací možnost "použít jazyk zařízení". V tom případě je jazyk aplikace automaticky nastavený podle jazyku daného zařízení. Pokud není jazyk zařízení aplikací podporován, je při tomto nastavení zobrazeno upozornění a jazyk aplikace je nastaven na defaultní - v tomto případě angličtina. Pokud uživatel nastaví aplikaci na nepodporovaný jazyk zařízení v runtime, je také zobrazeno vyskakovací upozornění. Stránku nastavení si lze prohlédnout na obr. 4.5.



Obr. 4.5: DiplomApp, obrazovka nastavení, platformy Android a iOS (vpravo).

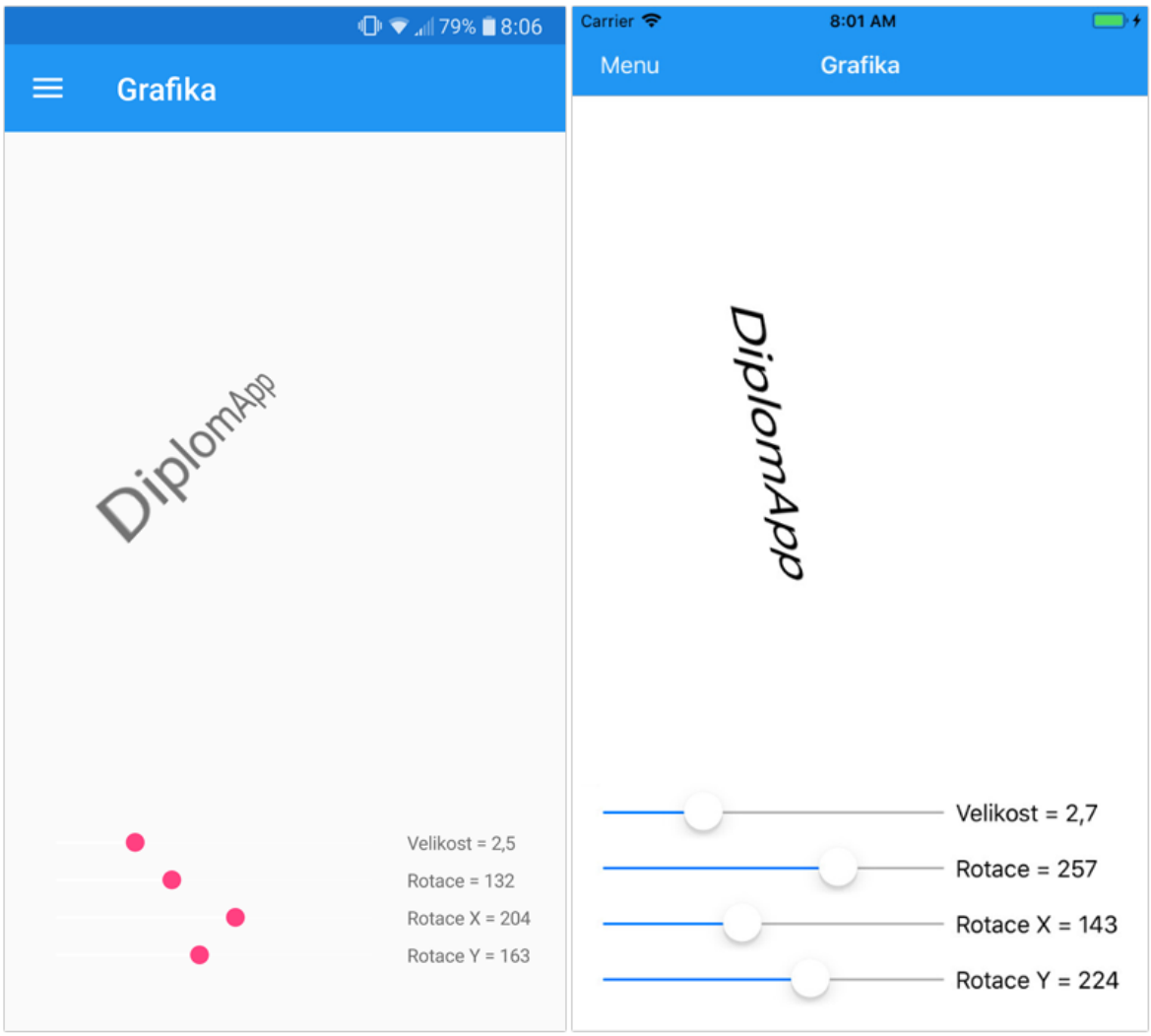
Pro uložení uživatelského nastavení aplikace je využito perzistivní ukládání vlastností aplikace. Tato možnost je vhodná pro uložení jednoduchých nastavení a informací o aplikaci, nikoliv však pro rozsáhlá uživatelská data. Při inicializaci aplikace jsou tato nastavení přečtena (pokud existují) a kultura aplikace se nastaví dle nich. Uložení jazykového nastavení vypadá následovně:

```
1 Application.Current.Properties["Language"] = SelectedLanguage;          // uložení vybraného jazyka
2 Application.Current.Properties["DeviceLanguage"] = UseDeviceLanguage;   // použití jazyka zařízení
3 Application.Current.SavePropertiesAsync();
```

4.2.4 Responzivní grafika uživatelského rozhraní

Návrh uživatelského rozhraní pomocí Xamarin.Forms v jazyce XAML dokáže zprostředkovat responzivní provázání vlastností jednotlivých použitých prvků. Pomocí datového provázání *Binding* lze propojit vlastnosti UI prvků nejen s funkcionalitou ve *ViewModel*, ale lze také propojit jednotlivé prvky přímo mezi sebou.

DiplomApp obsahuje stránku "Grafika" sloužící jako jednoduchý příklad provázání jednotlivých prvků uživatelského rozhraní. Na této stránce lze měnit vlastnosti (rotace a velikost) textu pomocí posuvníků. Veškerá logika je vytvořena pouze pomocí provázaných vlastností, tato stránka není obsluhována žádným funkčním kódem v jazyce C# . Stránka obsahuje čtyři posuvníky, které jsou provázány s popiskem obsahujícím daný text "DiplomApp". Každý z těchto posuvníků mění jinou vlastnost popisku: velikost a rotaci ve třech osách. Vedle každého posuvníku je poté další popisek. Tento popisek je přiřazený k posuvníku, vedle kterého se nachází a kromě popisu funkce posuvníku zobrazuje také jeho číselnou hodnotu. Tento jednoduchý příklad je inspirován [25] a lze si jej prohlédnout na obr. 4.6.



Obr. 4.6: DiplomApp, obrazovka demonstrující responzivní grafiku UI pomocí datového provázání v kódu XAML, platformy Android a iOS (vpravo).

Základem pro provázání mezi jednotlivými prvky je jejich pojmenování. Pojmenování se provádí pomocí vlastnosti *x:Name* kde x je definováno jako třída stránky obsahující

celé UI (*x:Class="DiplomApp.Views.GraphicsPage"*). V následující ukázce kódu si lze prohlédnout vytvoření popisku s textem, posuvníku ovládajícího velikost textu tohoto popisku a druhý popisek zobrazující nastavenou hodnotu:

```
<Label x:Name="label"
       Text="DiplomApp" />

<Slider x:Name="scaleSlider"
        BindingContext="{x:Reference label}"
        Maximum="10"
        Value="{Binding Scale, Mode=TwoWay}" />

<Label BindingContext="{x:Reference scaleSlider}"
       Text="{Binding Value, StringFormat='Velikost = {0:F1}'}" />
```

Jak je zřejmé, provázání se tvoří pomocí parametru *BindingContext*, ve kterém se nastaví reference na druhý provázaný prvek a to využitím jména daného prvku. Pokud není v prvku použitý parametr *BindingContext* ale je využito *Binding* pro některou z vlastností, využívá se defaultní kontext nastavený na začátku zdrojového kódu stránky (tedy nastavení propojovacího kontextu do *ViewModel*). Posuvník "scaleSlider" vytvořený ukázkou kódu má tedy nastavené provázání s popiskem "label". Jeho hodnota je poté provázána s vlastností *Scale* (tedy velikost, měřítko) daného popisku. Vzhledem k obousměrnému provázání (*Mode="TwoWay"*) posuvník nejen získává hodnotu z popisku, ale dokáže tuto hodnotu také měnit. Použitím posuvníku tak měníme velikost textu a tato změna se projevuje responzivně během runtime. Stejně tak je druhý popisek, který již není nijak pojmenován (není na něj navázán žádný jiný prvek - nepotřebuje jméno) provázán s posuvníkem. Využívá hodnotu posuvníku a tu zobrazuje jako svůj text.

Tímto jednoduchým principem lze vytvořit komplexní a hlavně responzivní uživatelská rozhraní, kde spolu jednotlivé prvky aktivně interagují. Tento způsob je využíván například při tvorbě prvků UI reagujících na proměnnou velikost prostoru, do kterého se musejí vykreslovat v závislosti na jiných zobrazených prvcích. Nevýhodou je změna kontextu k provázání pro daný prvek, nelze tedy využít provázání na další prvek UI a datové provázání do *ViewModel* najednou.

4.2.5 Lokálně ukládaná uživatelská data

U většiny aplikací potřebujeme uložit nějaká data lokálně. Pro jednoduché databáze je v mobilních aplikacích velmi využívaná technologie SQLite. Databáze SQLite mají velkou podporu v .NET, jsou tedy jasnou volbou i pro použití v aplikaci vyvíjené v Xamarin. V řešení aplikace DiplomApp byl využita implementace SQLite dostupná v NuGet balíčku SQLiteNetExtensions. Tento balíček nejen obsahuje .NET API pro práci s těmito databázeni, ale jak napovídá jeho název, obsahuje také rozšiřující funkce. Důležité rozšíření, které není v základní knihovně SQLite dostupné je přímá podpora databází typu one-to-many a many-to-many. Tyto databáze využívají rozdílnou architekturu propojování více

tabulek v databázi. One-to-many umožňuje přidat jako položku tabulky další tabulku, many-to-many pak umožňuje komplexní propojení více tabulek.

V DiplomApp je databáze využívána pro ukládání dat naměřených teplot, které aplikace získává z webové služby. Pro databázi byla zvolena architektura one-to-many. Jak bylo popsáno v kapitole 4.2.2, naměřená teplotní data získaná z webové služby jsou uložena do seznamu modelových objektů *TempItem*. Pro uložení těchto dat do databáze byl vytvořen další model *TempList*. Podoba tohoto modelu:

```
1 [PrimaryKey, AutoIncrement]
2 public int Id { get; set; }
3
4 public DateTime TimeSaved { get; set; }
5
6 [OneToMany]
7 public List<TempItem> TempDataList { get; set; }
```

[PrimaryKey] definuje primární klíč při ukládání těchto dat do databáze, tímto primárním klíčem je tedy proměnná int *Id*. *[AutoIncrement]* zajišťuje automatické inkrementování tohoto primárního klíče při ukládání do databáze, čímž se předchází konfliktu a přepisování uložených dat. Dále tato třída obsahuje proměnnou *TimeSaved* používanou k zaznamenání času uložení. Poslední položkou je seznam objektů *TempItem* jménem *TempDataList*. Tento seznam obsahuje samotná naměřená data, která chceme uložit. Použití *[OneToMany]* je velmi důležité pro správnou funkci propojování tabulek v databázi. Důležitou součástí objektu *TempItem* je kromě definovaného *[PrimaryKey]* také *[ForeignKey]*, který zajišťuje propojení se správným rodičem (objektem *TempList*). *TempItem* tedy musí obsahovat:

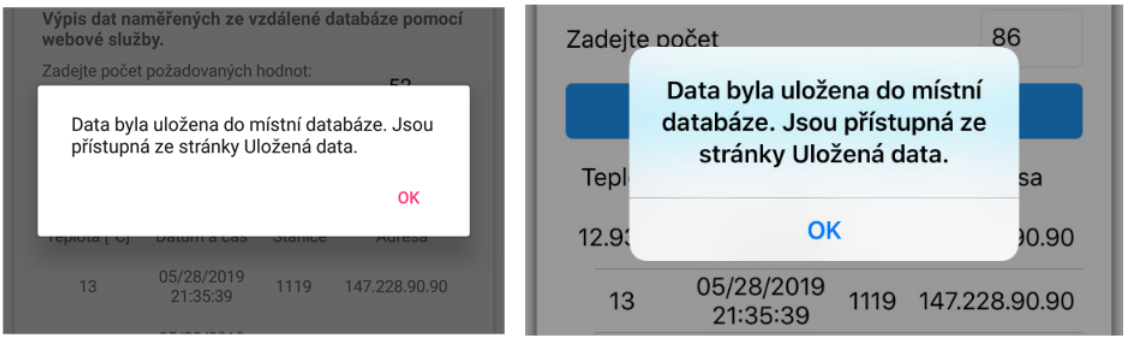
```
1 [PrimaryKey, AutoIncrement]
2 public int Id { get; set; }
3
4 ...
5
6 [ForeignKey(typeof(TempList))]
7 public int TempListId { get; set; }
```

Práce s databázemi pomocí SQLiteNetExtensions je velmi jednoduchá. Základem je definování cesty, na které bude databáze vytvořena a kam se k ní bude přistupovat. Vzhledem k vytváření lokální databáze bude tato cesta jednoduše určovat polohu databáze v zařízení. Definování cesty k lokální databázi:

```
1 private string _dbPath = Path.Combine(System.Environment.GetFolderPath(Environment.SpecialFolder.Personal),
2                                       "TempDB.db3");
```

Ukládání dat poté vypadá následovně. Nejprve je vytvořena nová položka *TempList*, do které se zaznamená aktuální čas (čas ukládání dat). Dále se vytvoří nové připojení k

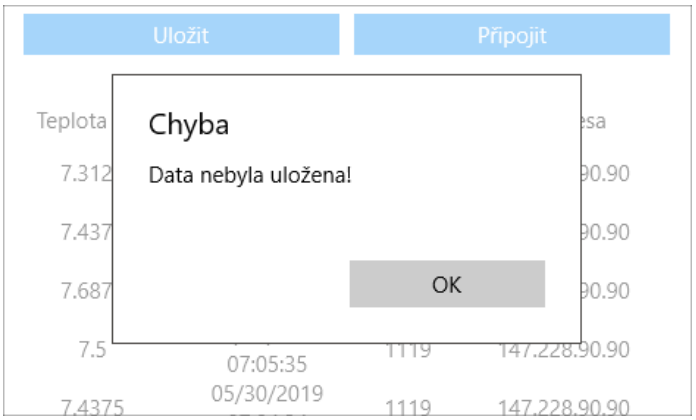
databázi do proměnné *db* (viz následující ukázky kódu) a tato proměnná je poté využívána k veškeré práci s danou databází. V databázi se vytvoří dvě nové tabulky - jedna pro *TempItem* a jedna pro *TempList*. Pomocí cyklu jsou uloženy všechny položky *TempItem* v aktuálním seznamu hodnot získaných z webové služby (seznam *Temp*). Následně je do tabulky zapsán také nový záznam *TempList*, je mu přiřazen seznam položek *TempItem* a pomocí *UpdateWithChildren()* je vytvořeno propojení těchto dvou tabulek. Nakonec je odeslána zpráva o uložení dat, která je přijímána v zobrazované stránce, kde má za následek vyskočení dialogového okna informujícího o uložení (obr. 4.7). Popsanou implementaci si lze prohlédnout v následující ukázce kódu.



Obr. 4.7: DiplomApp, dialogové okno informující o uložení dat do databáze, platformy Android a iOS (vpravo).

```
1  var newEntry = new TempList()
2  {
3     TimeSaved = DateTime.Now,
4  };
5
6  var db = new SQLiteConnection(_dbPath);
7  db.CreateTable<TempList>();
8  db.CreateTable<TempItem>();
9
10 foreach (var item in Temp)
11 {
12     db.Insert(item);
13 }
14
15 db.Insert(newEntry);
16 newEntry.TempDataList = Temp;
17 db.UpdateWithChildren(newEntry);
18
19 MessagingCenter.Send(this, "DataSaved");
```

Funkce pro uložení dat je obsažena v *ConnectivityViewModel* a je spouštěna asynchronně ve vlastním vláknu separátně od vlákna UI. UI této stránky je navrženo tak, že je tlačítko pro uložení dat funkční pouze po úspěšném získání dat z webové služby. Uložením dat se toto tlačítko znefunkční až do nahrání nových dat (jednoduché zamezení ukládání duplicitních souborů dat). Tato funkce je také ošetřena pomocí *try - catch* a při výskytu chyby během ukládání do databáze je pomocí *MessagingCenter* vyvoláno informující dialogové okno (obr. 4.8).



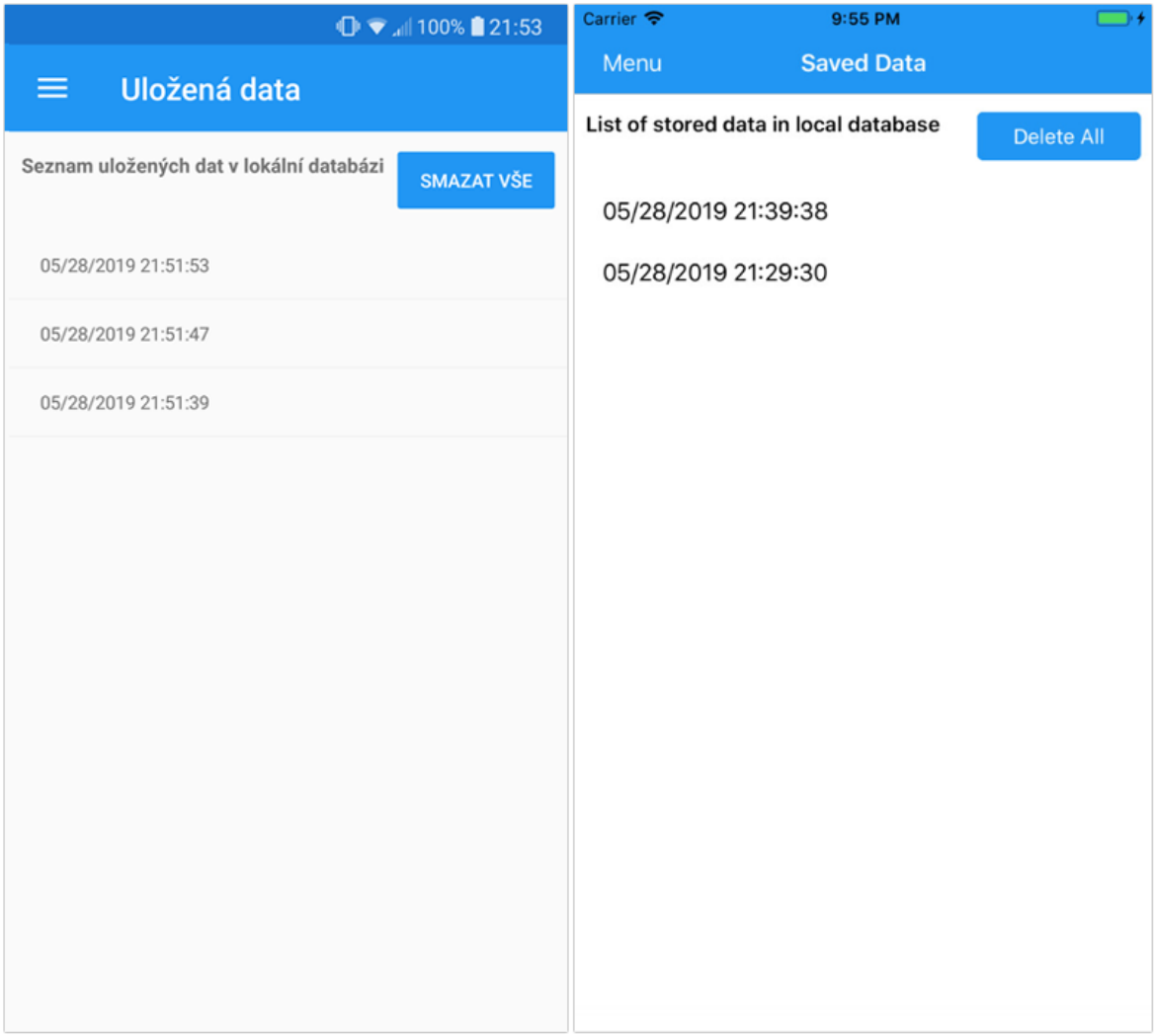
Obr. 4.8: DiplomApp, dialogové okno upozorňující na výskyt chyby během ukládání dat do databáze, platforma Windows.

Pro zobrazení dat uložených v databázi je vytvořena nová stránka *SavedData*. *ViewModel* této stránky obsahuje funkci pro načtení dat z databáze. Připojení do databáze je vytvořeno stejným způsobem jako při ukládání databáze. Následuje načtení tabulky obsahující všechny záznamy typu *TempList* do seznamu *TempSaved*. Toto jednoduché načtení celého obsahu tabulky včetně seřazení podle *Id* má ovšem tu nevýhodu, že nenačte přidruženou tabulku obsahující položky *TempItem*. K tomu je využito jednoduchého cyklu, který pomocí Id přiřadí ke každému objektu *TempList* správný seznam objektů *TempItem*:

```
1 var db = new SQLiteConnection(_dbPath);
2 TempSaved = db.Table<TempList>().OrderByDescending(x => x.Id).ToList();
3
4 foreach (var item in TempSaved)
5 {
6     var _TempValuesStored = db.GetWithChildren<TempList>(item.Id);
7     item.TempDataList = _TempValuesStored.TempDataList;
8 }
```

Funkce pro načtení dat je také ošetřena pomocí *try - catch* proti možným problémům s databází. Dále ovládá zobrazení textu informujícího o prázdné databázi a také zpřístupňuje tlačítko pro vymazání databáze (zpřístupněné pouze pokud není prázdná). Stránka zobrazuje seznam uložených měření v elementu *listView*. Pro každou položku je zobrazen časový štítek, kdy byla tato data uložena. Dále obsahuje popisek stránky a zmíněné tlačítko pro vymazání všech dat. Pokud nejsou k dispozici žádná data, je zobrazen navíc také zmíněný popisek informující o prázdné databázi. Stránku si lze prohlédnout na obr. 4.9.

Při kliknutí na položku v seznamu se otevírá nové okno *TempItemDetailPage*, které zobrazuje vlastní uložená naměřená data. Tato funkce je vytvořena jiným způsobem než zbytek aplikace. Obsahuje ji totiž samotná stránka (*View*) a ne *ViewModel*. To umožňuje jednoduché vytvoření nové stránky pomocí *Navigation.PushAsync()*. Tuto novou stránku poté můžeme jednoduše také jejího objektu *View* zavřít pomocí *Navigation.PushAsync()*, což se bude dále hodit. Této funkcionality se dá dosáhnout i přes *ViewModel* pomocí



Obr. 4.9: DiplomApp, obrazovka - seznam uložených balíčků dat v databázi, platformy Android a iOS (vpravo).

delegátů, pro toto jednoduché použití je ale vhodnější tato struktura. Při kliknutí na položku v seznamu tedy vytvoříme novou stránku, které jako parametr předáme právě onu položku (tedy vybraný objekt *TempList* ze seznamu) Funkce vypadá takto:

```
async void OnItemSelected(object sender, SelectedItemChangedEventArgs args)
{
    var item = args.SelectedItem as TempList;
    if (item == null)
    return;

    await Navigation.PushAsync(new TempItemDetailPage(new TempItemDetailViewModel(item)));
    lvSaved.SelectedItem = null;
}
```

Model nově vytvořené stránky *TempItemDetailViewModel* má k dispozici daný objekt *TempList* jako svůj zdroj dat. Samotná stránka tedy pomocí *Binding* k dokáže k těmto datům přistoupit a zobrazit je. Za povšimnutí stojí hlavička této stránky. Vzhledem k metodě vytvoření této stránky je v levém rohu místo přístupu do menu tlačítko pro zavření této stránky (podstránky) a návrat na stránku *SavedDataPage*. Jako titulek stránky je použit časový štítek z objektu *TempList*, v pravém rohu je poté tlačítko pro vymazání. Samotná stránka má poté stejnou strukturu jako přímé zobrazení dat na stránce *ConnectivityPage* obsluhující webové služby. Tabulka obsahující popisek tabulky dat a pod ní seznam zobrazující veškeré položky *TempItem* obsažené v poskytnutém objektu *TempList*. Stránku si lze prohlédnout na obr. 4.10.

Tlačítko "Smazat" není obsluhováno přes *ViewModel* ale v objektu *View* a to z již zméněného důvodu, kdy se dá tato stránka jednoduše zavřít pomocí *PopAsync()*. Funkce tohoto tlačítka je tedy realizována přímo v objektu *TempItemDetailPage*. Jak text tlačítka napovídá, jedná se o smazání tohoto záznamu z databáze. Implementace vypadá následovně:

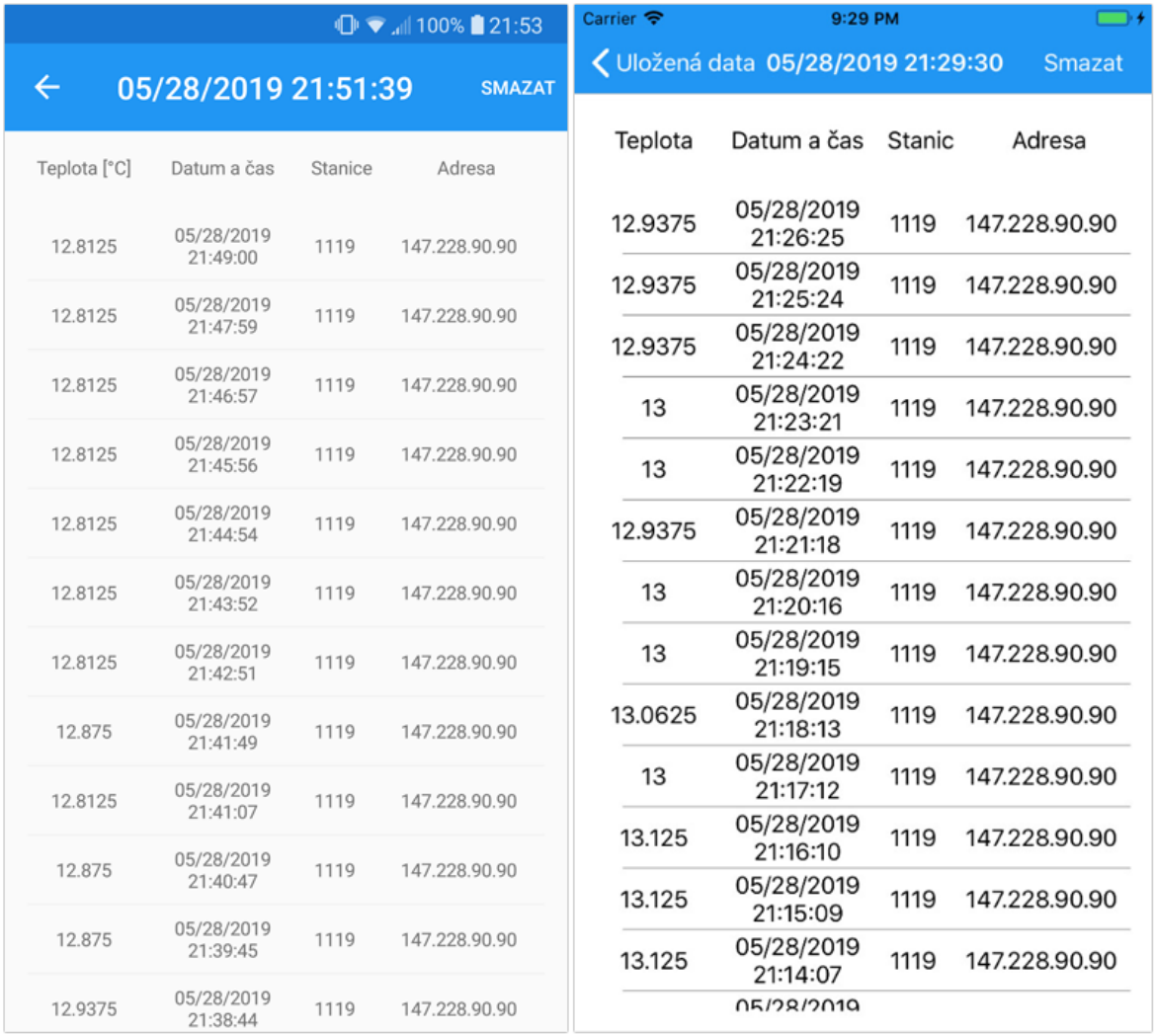
```
private async void BtnDelete_Clicked(object sender, EventArgs e)
{
    var db = new SQLiteConnection(_dbPath);

    var item = db.Get<TempList>(viewModel.TempListData.Id);
    db.GetChildren(item, true);
    db.Delete(item, recursive: true);

    MessagingCenter.Send(viewModel, "DataDeleted");

    await Navigation.PopAsync();
}
```

Kvůli možnosti mazání záznamu z databáze má tato stránka také svůj databázový přístup. Do proměnné *item* je přiřazena stejná položka *TempList*, které tato stránka patří. Aby bylo zajištěno smazání nejen dané položky *TempList* ale i jejího seznamu *TempItem* dětí, je třeba tyto položky načíst pomocí *GetChildren*. Poté lze pomocí *db.Delete* smazat



Obr. 4.10: DiplomApp, obrazovka - detail dat uložených v databázi, platformy Android a iOS (vpravo).

vše za předpokladu nastaveného parametru *recursive*. Ve funkci je dále odeslána zpráva, která zapříčiní obnovení dat z databáze do zobrazeného seznamu na stránce *SavedDataPage*. Na tuto stránku se následně aplikace vrací pomocí *await Navigation.PopAsync()*. Pokud by nebylo zavoláno obnovení dat na této stránce, obsahovala by stále i právě smazanou položku.

Nakonec zbývá zmínit onu funkci "Smazat vše" na stránce *SavedDataPage*. Implementace je velmi podobná předchozí funkci pro smazání jedné položky. Nejdříve je do seznamu *TempSaved* obnoven stav databáze a poté je pomocí cyklu provedeno pro každou jednotlivou položku načtení dítěte - seznamu *TempItem*. Pomocí rekurzivního mazání je poté tato položka kompletně smazána. Nakonec je zavoláno obnovení dat pro UI. Tato funkce je také ošetřena pomocí try - catch např. pro případ pokusu o mazání prázdné databáze. Implementaci si lze prohlédnout zde:

```
1 var db = new SQLiteConnection(_dbPath);
2 TempSaved = db.Table<TempList>().OrderByDescending(x => x.Id).ToList();
3
4 foreach (var item in TempSaved)
5 {
6     db.GetChildren(item, true);
7     db.Delete(item, recursive: true);
8 }
9 LoadData();
```

5

Závěr

Jedním z cílů této práce bylo prozkoumání využití mobilních aplikací a jejich vývoj. Využití mobilních aplikací bylo představeno z pohledu dostupných technologií moderních zařízení. Za popularitou těchto zařízení a zprostředkovaně také obsažených aplikací stojí technologický pokrok. Každé moderní mobilní zařízení je výkonný počítač obsahující připojení k internetu, služby polohy a v neposlední řadě stále také umožňuje přímý kontakt skrze hlasové hovory. Díky dotykovému rozhraní jsou tato zařízení velice uživatelsky přívětivá a našla si cestu prakticky do každé kapsy. Velkým omezením ovšem zůstává technologie akumulátorů, která je momentálně hlavním limitujícím faktorem mobilních zařízení.

Hlavním cílem této práce bylo vytvoření mobilní aplikace navázané na skutečná data projektů katedry KAE. Tato aplikace pojmenovaná DiplomApp byla vytvořena jako multiplatformní aplikace pro nejrozšířenější mobilní platformy současnosti. Těmito platformami jsou mobilní operační systémy Android a iOS. Vyvíjená byla také verze aplikace pro operační systém Windows, který již ale opustil aktivní působení mezi mobilními zařízeními. Této verzi aplikace byl tedy přikládán menší důraz.

DiplomApp byla vyvinuta na několik mobilních platforem pomocí vývojového frameworku Visual .NET Xamarin. Tento framework spadající pod .NET vlastněný společností Microsoft v současnosti nabízí pravděpodobně nejlepší volbu pro vývoj aplikace pro několik platforem. Podporuje vývoj pro všechny nejrozšířenější platformy a s jeho využitím lze dosáhnout až k 97% mobilních zařízení na trhu. Veškerý vývoj aplikace pod Xamarin probíhá v jazyce C# s využitím rozsáhlých knihoven .NET. Odpadá tedy vývoj jednotlivých verzí stejné aplikace v různých jazycích a navíc umožňuje rozsáhlé sdílení kódu mezi jednotlivými platformami.

Aplikace obsahuje několik funkcí, na jejichž vývoji a zdrojovém kódu je v práci popisován vývoj ve frameworku Xamarin. Jako zdroje dat pro aplikaci byly využity webové služby poskytnuté katedrou KAE jako výukový projekt. Tyto služby poskytují naměřená teplotní data ze stanice umístěné na katedře. DiplomApp tato data zpracovává a

umožňuje jejich přehledné zobrazení. Aplikace také dokáže tato data přehledně ukládat do lokální databáze SQLite. Z této databáze jsou naměřená data v aplikaci jednoduše spravována a kdykoliv přístupná i bez momentálního připojení k síti. Pro demonstraci využívání různých zdrojových souborů jsou v aplikaci implementovány dva jazyky uživatelského rozhraní, mezi kterými může uživatel libovolně přepínat. Aplikace také umožňuje nastavení, kdy se samotná aplikace automaticky přepne do odpovídajícího jazyka podle jazyka zařízení. Pro demonstrování možností tvorby responzivního uživatelského rozhraní obsahuje aplikace také stránku s jednoduchým příkladem, ve kterém může uživatel upravovat vlastnosti grafického prvku a živě sledovat projevy těchto změn.

V práci tedy bylo popsáno využití aplikací a byla úspěšně vyvinuta vzorová aplikace. Prostor pro rozšíření a pokračování ovšem je. Práce by se dala rozšířit implementací dalších služeb a využitím dalších mobilních technologií ve vzorové aplikaci. Z časových důvodů byl implementován pouze omezený počet vzorových funkčností. Vzhledem k menšímu důrazu na vývoj této aplikace na platformu Windows není bohužel zaručena kompletní funkčnost této verze aplikace. DiplomApp pro Android a iOS obsahuje veškerou funkčnost popsanou výše, verze pro Windows bohužel neobsahuje správně fungující ukládání dat do lokální databáze SQLite.

Řešení DiplomApp obsahuje několik neodstraněných varování. Windows projekt obsahuje varování týkající se certifikátů potřebných pro publikování aplikace, které nebyly vytvořeny a přiřazeny k projektu. Vzhledem k nepublikování aplikace jsou tato varování ignorována. Varování chybějícího defaultního jazyku je také možno ignorovat, vzhledem k nesedící nomenklatuře en vs en-US, lokalizace ale funguje. Projekty Android a iOS obsahují varování týkající se neaktuálního frameworku Xamarin (nesedící verze různých knihoven). Toto je daň za nevyužití nejnovější verze Xamarin, která není funkční s použitým starším systémem macOS. Jako poslední je možný výskyt chyby ResXFileCodeGenerator. Tato chyba je pouze známý bug IDE Visual Studio a neovlivňuje funkčnost používání resource zdrojových souborů při lokalizaci.

Literatura

[1] Siuhi, S., Mwakalonge, J., 2016 *Opportunities and challenges of smart mobile applications in transportation* Department of Civil Engineering, Abu Dhabi University, Abu Dhabi, United Arab Emirates, Department of Civil and Mechanical Engineering Technologz and Nuclear Engineering, South Carolina State University, Orangeburg, SC 29117, USA

[2] Bates, S., 2014 *A History of Mobile Application Development* Dostupné z: https://manifesto.co.uk/history-mobile-application-development/

[3] Rajput, M., 2015 *Tracing the History and Evolution of Mobile Apps* Dostupné z: https://tech.co/news/mobile-app-history-evolution-2015-11

[4] Shao, S., Wang, E., *Die Photo Analysis* Dostupné z: http://vlsiarch.eecs.harvard.edu/research/accelerators/die-photo-analysis/

[5] Priya, L., Visalaxi, S., Mohana, E., 2016 *A Study on Smart Phone Batteries and its Power Utilization* Department of Information Technology, Rajalakshmi College of Engineering, Chennai, India

[6] Kor, ah-lian, 2017 *Energy Consumption in Smartphones: An Investigation of Battery and Energy Consumption of Media Related Applications on Android Smartphones* School of Computing, Creative Technologies and Engineering, Leeds Beckett University, Leeds, United Kingdom

[7] *Official Samsung News Channel regarding Galaxy Note 7 Device* Dostupné z: https://news.samsung.com/us/tag/galaxy-note7/

[8] Sanchez, M. I., de la Oliva, A., Bernardos, C. J., 2016 *How does my smartphone manage network connections?* IMDEA Networks Institute, Spain, Departmento de Ingenieria Telematica, Universidad Carlos III de Madrid, Spain

[9] Dube, R., 2018 *Capacitive vs. Resistive Touchscreens: What Are the Differences?* Dostupné z: https://www.makeuseof.com/tag/differences-capacitive-resistive-touchscreens-si/

[10] Krithikaa, M., 2016 *Touch Screen Technology – A Review* IT Department, Sri Krishna Arts and Science College, Coimbatore, India

[11] 2019 *Mobile and Tablet Market Share Worldwide - StatCounter Global Stats* Dostupné z: http://gs.statcounter.com/os-market-share/mobile-tablet/worldwide/#monthly-201804-201904-bar

[12] Apple, 2019, *Apple Developer* Dostupné z: https://developer.apple.com/

[13] Apple, 2019, *XCode - Apple Developer* Dostupné z: https://developer.apple.com/xcode/

[14] 2019 *Dokumentace pro Xamarin - Xamarin — Microsoft Docs* Dostupné z: https://docs.microsoft.com/cs-cz/xamarin/

[15] *Xamarin Live Player for Xamarin Forms* Dostpuné z: https://docs.microsoft.com/cs-cz/xamarin/tools/live-player/

[16] Prajapati, M., Phadake, D., Poddar, A., 2016 *Study on Xamarin Cross-Platform Framwork* Department of MCA, B. Tech. - Information Technology, University of Mumbai, Manipal University Jaipur, Mumbai, India

[17] Haris, M., Jadoon, B., Khan, F. H., 2017 *Evolution of Android Operating System: A Review* Department of Computer Science, COMSATS Institute of Science and Technology, Islamabad, Pakistan Department of Computer Science, Federal Urdu University of Arts, Sciences and Technology, Islamabad, Pakistan Knowledge and Data Science Research Center, Department of Computer Engineering, College of EME, National University of Sciences and Technology, Islamabad, Pakistan

[18] Xamarin Inc., 2015 *Key Approaches for Mobile Success* Dostupné z: http://cdn1.xamarin.com/resources/xamarin-white-paper-key-approaches-to-mobile-success.pdf

[19] Taft, D., K., 2016 *Microsoft Makes Xamarin free in Visual Studio, Open-Sources SDK* Dostupné z: https://www.eweek.com/development/microsoft-makes-xamarin-free-in-visual-studio-open-sources-sdk

[20] Microsoft, 2018 *Přehled sdílení kódu - Xamarin — Microsoft Docs* https://docs.microsoft.com/cs-cz/xamarin/cross-platform/app-fundamentals/code-sharing

[21] Microsoft, 2018 *Architektura aplikací Android - Xamarin — Microsoft Docs* Dostupné z: https://docs.microsoft.com/cs-cz/xamarin/android/internals/architecture

[22] Microsoft, 2018 *Architektura aplikací iOS - Xamarin — Microsoft Docs* Dostupné z: https://docs.microsoft.com/cs-cz/xamarin/ios/internals/architecture

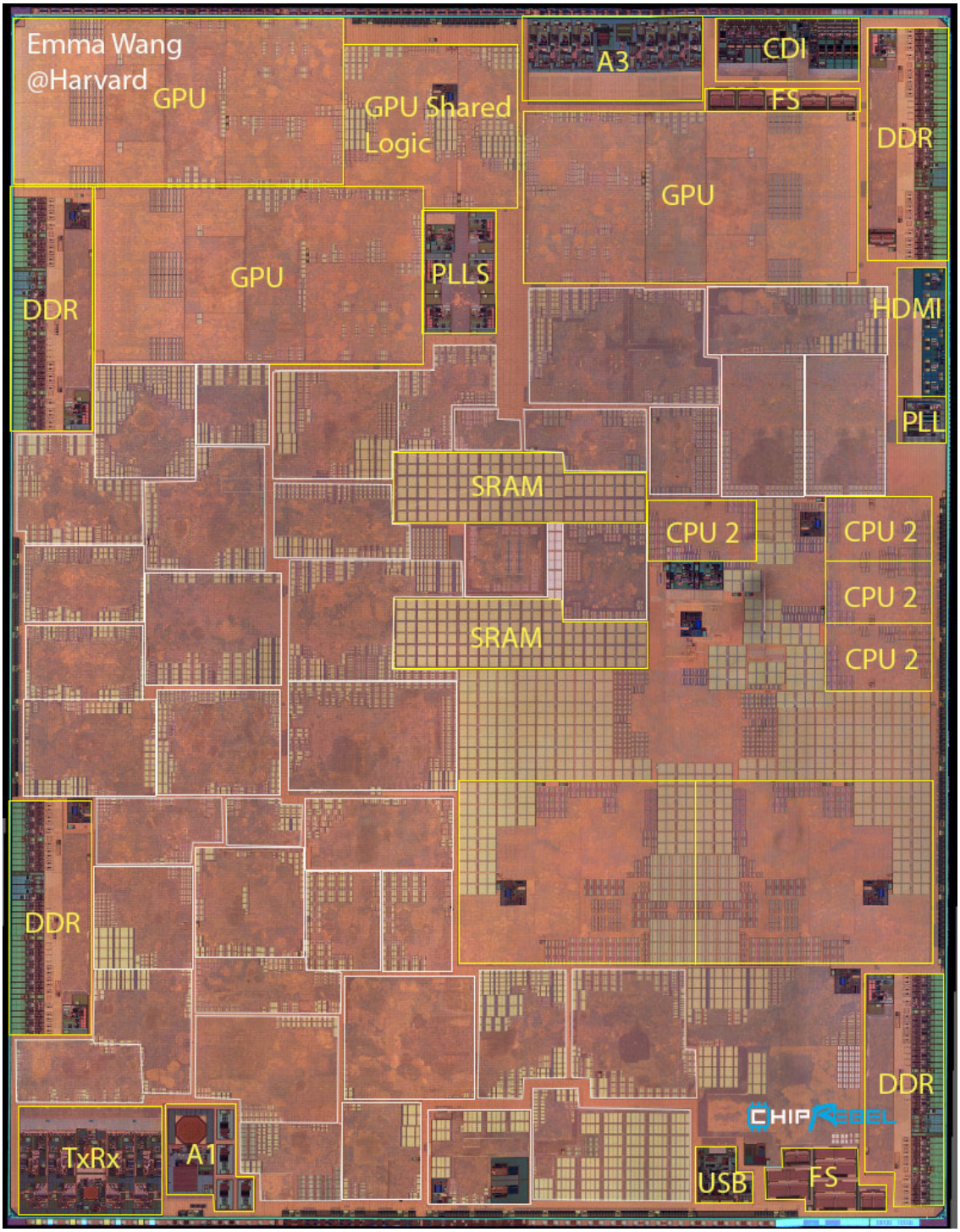
[23] *LogMeIn Hamachi* Software pro tvorbu virtuálních privátních sítí. Dostupné z: https://www.vpn.net/

[24] Nijs, P., 2017 *Dynamically binding RESX Resources in Xamarin Forms* Dostupné z: https://blog.pieeatingninjas.be/2017/05/20/dynamically-binding-resx-resources-in-xamarin-forms/

[25] Microsoft, 2018 *Část 4. Základy vytváření vazeb dat - Xamarin — Microsoft Docs* Dostupné z: https://docs.microsoft.com/cs-cz/xamarin/xamarin-forms/xaml/xaml-basics/data-binding-basics

Příloha A

Obrázky



Obr. A.1: Analýza struktury SoC z mobilního zařízení od firmy Apple. Převzato z: [4].