Nástroj pro správu nemovitostí pro malé investory

Jan Kurfürst

Bakalářská práce
2022

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

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

Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Jan Kurfürst**
Osobní číslo:	**A19057**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Nástroj pro správu nemovitostí pro malé investory**
Téma práce anglicky:	**Property Management Tool for Small Investors**

Zásady pro vypracování

1. Popište současný stav technologií pro vývoj a zabezpečení multiplatformních aplikací.
2. Zvolte a popište vhodný framework pro implementaci aplikace.
3. Navrhněte aplikaci s využitím popsaných technologií, popište případy použití a architekturu aplikace.
4. Navrhněte způsob zabezpečení aplikace.
5. Realizujte vývoj navržené aplikace a popište její klíčové části.
6. Demonstrujte výsledky a formulujte závěr.

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

Seznam doporučené literatury:

1. .NET documentation. Microsoft Docs [online]. Oficiální dokumentace firmy Microsoft Corporation [cit. 2021-10-12]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/
2. SHELDON, Amyra. 11 Popular Cross-Platform Tools for App Development (Updated Version 2021). Hackernoon.com [online]. Edwards, Colorado: ARTMAP, 2016, March 20th 2020 [cit. 2021-10-15]. Dostupné z: https://hackernoon.com/9-popular-cross-platform-tools-for-app-development-in-2019-53765004761b
3. PANIGRAHY, Nilanchala. Xamarin Mobile Application Development for Android: Develop, test, and deliver fully featured Android applications using Xamarin. Second Edition. Birmingham B3 2PB, UK: Packt Publishing, August 2015. ISBN 9781785280375.
4. HERMES, Dan a Nima MAZLOUMI. Building Xamarin.Forms Mobile Apps Using XAML: Mobile Cross-Platform XAML and Xamarin.Forms Fundamentals. California: Apress, 2019. ISBN 978-1-4842-4029-8.
5. HERMES, Dan. Xamarin Mobile Application Development: Mobile Cross-Platform XAML and Xamarin.Forms Fundamentals. California: Apress, 2015. ISBN 9781484202159.

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

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

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

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

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

Ve Zlíně dne 24. ledna 2022

Prohlašuji, že

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

Prohlašuji,

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

Ve Zlíně, dne 16.05.2022

Jan Kurfürst, v. r.
podpis studenta

ABSTRAKT

Bakalářská práce „Nástroj pro správu nemovitostí pro malé investory“ se zaměřuje na současné technologie pro vývoj multiplatformních aplikací. V první teoretické části je popsán současný stav multiplatformních technologií a jejich rozdělení. Zaměřuje se tedy na všechny možnosti vývoje mobilní aplikace se zacílením na více platforem. V druhé teoretické části se práce zaměřuje na bezpečnost a zabezpečení mobilních aplikací. V této části jsou popsány možné hrozby a slabiny, kterých mohou útočníci využít, a zároveň jsou zde uvedeny techniky, jak aplikace zabezpečit, aby byl útok co nejspolehlivěji znemožněn. Praktická část se zaměřuje na vývoj aplikace s pracovním názvem „Realtys“, na které je demonstrována nová technologie od společnosti Microsoft .NET MAUI. V této části je popsána aplikace a její klíčové části, kdy je zaměření výhradně na prvky, které tato nová technologie přináší do světa multiplatformního vývoje aplikací. Tyto poznatky a zkušenosti jsou pak sepsány v závěru práce.

Klíčová slova: .NET MAUI, Xamarin, C#, Multiplatformní, Nativní, IOS, Android

ABSTRACT

The bachelor thesis "Asset Management Tool for Small Investors" focuses on current technologies for the development of multiplatform applications. The first theoretical part describes the current state of multiplatform technologies and their distribution. It therefore focuses on all possibilities of developing a mobile application with a focus on multiple platforms. The second theoretical part is focused on the safety and security of mobile applications. This describes possible threats and vulnerabilities that attackers can exploit and how applications can prevent an attack in the most reliable way possible. The practical part is focused on the development of an application with the working name "Realtys", which demonstrates a new technology from Microsoft .NET MAUI. This section describes the application and its key parts, focusing exclusively on the elements that this new technology brings to the world of cross-platform application development. These findings and experiences are then written at the end of the work.

Keywords: : .NET MAUI, Xamarin, C#, Cross-platform, Native, IOS, Android

Děkuji své přítelkyni a rodině za konstantní podporu a vytvořené zázemí při zhotovování této práce. Dále bych chtěl poděkovat Ing. Eriku Královi, Ph.D. za profesionální rady a čas, který věnoval svým studentům.

Dělej jen to, co tě opravdu baví.

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

OBSAH

ÚVOD

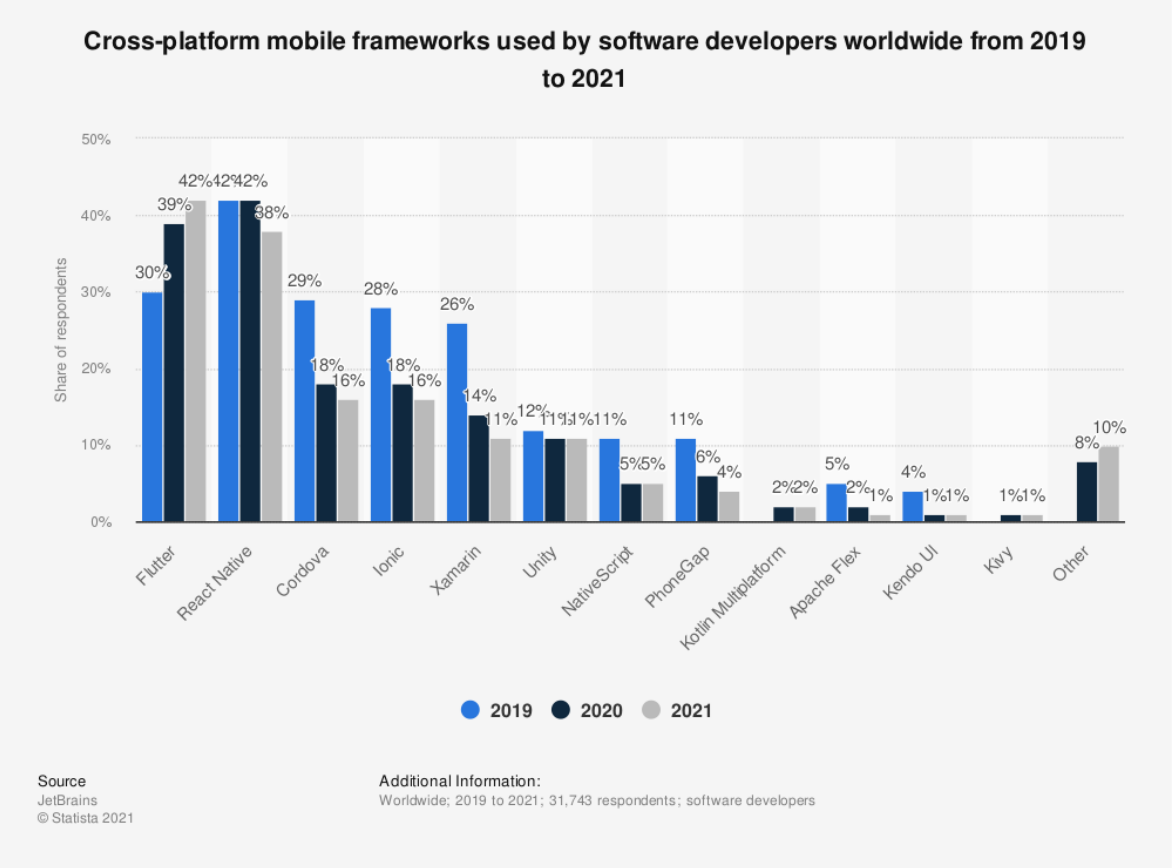
Bakalářská práce se zabývá vývojem multiplatformních aplikací a současnými technologiemi na ně zaměřenými. Multiplatformní vývoj se v dnešní době daleko více rozvíjí a mnoho firem i jednotlivců, kteří chtějí obsáhnout širší spektrum uživatelů a přitom zohlednit i cenu vývoje jejich aplikace, mnohem častěji upřednostní kvalitní multiplatformní aplikaci, než dvě a více stejných aplikací zvlášť na všechny platformy, jenž využívá jejich cílová skupina. Tím toto univerzální řešení nabírá na síle na trhu. Přesto nejsou vývoj a výsledná aplikace vždy dokonalé a vývojář musí znát také techniky specifické pro dané platformy. V mnoha různých technických článcích je možné se dočíst o spoustě technologiích, které již existují. Každá má ovšem v sobě nějaký háček, ať už je to zaměření pouze na mobilní zařízení nebo nemožnost využít veškeré systémové funkce cílové platformy. Také výkon aplikací nemusí být vždy optimální. Co se týče zabezpečení, tak každá aplikace ošetřuje zabezpečení jinak, v závislosti na zvolené technologii.

Práce je členěna na část praktickou a část teoretickou. Rozdělena je do osmi kapitol, kde první tři patří do teoretické části a dalších pět k části praktické. Poslední kapitola praktické části shrnuje technologii .NET MAUI. Na začátku jsou sepsány všechny druhy vývoje aplikací na více platforem. Jsou zde zmíněni i konkrétní zástupci těchto technologií, kteří se hojně využívají při tvorbě aplikací. Dále je popsáno, jaké hrozby ohrožují tyto aplikace a jak se jim co nejlépe bránit. V praktické části je pak popsán vývoj aplikace v nové technologii od společnosti Microsoft .NET MAUI, která umožňuje vývoj multiplatformní aplikace jak na mobilní, tak i na desktopová zařízení. Cílem práce je tedy předání poznatků a zkušeností s vývojem pomocí této nové technologie a přiblížení, co nového přináší oproti konkurenčním technologiím.

I. TEORETICKÁ ČÁST

1 SOUČASNÉ TECHNOLOGIE PRO VÝVOJ MULTIPLATFORMNÍCH MOBILNÍCH APLIKACÍ

V současné době existuje mnoho metod, frameworků, nástrojů a technologií pro vývoj mobilních aplikací. Mimo mobilní zařízení už některé multiplatformní technologie podporují i desktopová zařízení. Technologie se obecně rozděluje na skupiny podle nástrojů pro jejich vývoj. Mezi tyto skupiny patří PWA, hybridní aplikace, multiplatformní aplikace a nativní aplikace.[19] Každá z těchto skupin používá rozdílné metody a frameworky a má různé výhody a nevýhody související s jejich technologiemi. [22]



Obrázek 1. Multiplatformní frameworky používané vývojáři 2019-2021 [14]

Podle průzkumu od JetBrains, zprostředkovaným společností Statista (viz. Obrázek 1), lze vidět popularitu jednotlivých technologií mezi softwarovými vývojáři. Je zde vidět oblíbenost, jak u hybridních aplikací, tak u čistě multiplatformních (bez použití webových technologií) v průběhu posledních tří let. [14]

1.1 Význam pojmu multiplatformní programování

Pojmem „multiplatformní programování“ je vystižena metoda programování, kdy si vývojář napíše logiku aplikace pouze jednou, a přitom je kód spustitelný na různých platformách jak už na mobilních (Android, iOS), tak i desktopových zařízeních s rozdílnými operačními systémy jako Windows, Linux i na macOS. Tato metoda má v IT komunitě i svůj vlastní název a tím je „*Write once, run anywhere*“ známější spíše pod zkratkou WORA (někdy i WORE), což ve volném překladu znamená „Napsat jednou, rozběhnout kdekoliv“. Slouží to především jak k usnadnění pro vývojáře, tak i k ušetření financí na vývoji, kdy prakticky stačí napsat aplikaci pouze jednou, a přesto zahrnout širší skupinu zařízení pro nasazení. [1]

Dnes je již jednoduché najít technologie a nástroje pro takovýto způsob vývoje se širokou škálou možností. Díky tomuto množství technologií je k vývoji zahrnuto i spoustu vyhledávaných a mezi vývojáři oblíbených programovacích jazyků. Existují nástroje, které jsou založené na webových technologiích za použití TypeScriptu/JavaScriptu, CSS a Html. Stejně tak i technologie, které používají i nízkoúrovňové nebo objektově orientované jazyky, jako je C, C++, C# anebo Java. Samozřejmě znalost těchto jazyků je velkou výhodou, protože některé frameworky používají i moduly psané v těchto jazycích a to může být nejméně pomoc při vývoji multiplatformních aplikací. [2]

1.2 Význam pojmu multiplatformní aplikace

Multiplatformní aplikace jsou aplikace, které jsou zacílené na více zařízení s rozdílnými operačními systémy i překladači. Jsou to aplikace, jenž stačí vyvinout pouze jednou a za pomoci určitého frameworku, který odpovídá preferencím vývojáře, ať už zacílením, nebo programovací metodou či jazykem, ve kterém je zběhlý. Tyto aplikace budou fungovat na všech zařízeních, na které je daný framework nebo nástroj zaměřený.

Aplikace samozřejmě musí být nějakým způsobem zabezpečena a neměla by mít úplný přístup ke všem funkcím cílového systému, obzvláště k soukromým datům nebo neopodstatněnému využívání hardwaru. K tomu slouží různá omezení a povolení, která se dělí na různé typy. Google, pro zařízení s operačním systémem typu Android, tato povolení ve svých dokumentacích pro Android developery nazval *Install-time permissions, runtime permissions* a *special permissions*. [3]

Multiplatformní aplikace má jisté výhody, ale nezaručují bezproblémový vývoj, protože i přesto, že je aplikována technika WORA, některé části aplikace se musí napsat nativně pro

danou platformu. Samozřejmě čím novější technologie, tím se více eliminují tyto nežádoucí elementy a ubývá potřeby vyvíjet nativní části aplikace.

1.3 Nativní aplikace

Nativní aplikace je aplikace vyvinuta na konkrétní zařízení a jejich operační systém, ke které mají uživatelé přístup přes domovskou obrazovku. Tyto aplikace jsou obvykle získávány prostřednictvím App Store nebo Google Play. Rozdíl mezi nativní aplikací a webovou aplikací jsou především programovací jazyky. Zatímco webové aplikace jsou psány primárně v Javascriptu, nativní aplikace používají různé jazyky v závislosti na platformě, na které jsou postaveny. Například pro vývoj nativních aplikací pro Android se využívá Java nebo Kotlin, pro iOS je to pak Objective-C nebo Swift a pro Windows Phone většinou C#. Jakmile je aplikace nainstalována do zařízení, může získat přístup ke všem funkcím zařízení, jako je fotoaparát, GPS, akcelerometr, kompas, seznam kontaktů atd. Kromě toho mají nativní aplikace další možnosti, jako je schopnost začlenit gesta, a to buď standardní gesta operačního systému, nebo nová gesta definovaná aplikací. Také mohou používat oznamovací systém zařízení a pracovat offline. [15]

1.3.1 Výhody

Největší výhodou nativního vývoje je nejspíše vysoká kvalita aplikace, která je navržena pro konkrétní platformu, a to od optimalizace a rychlosti, přes ukládání a uchovávání dat, až po efektivní využití hardwaru zařízení. Není zde problém při programování složitějších aplikací, aby nějaká část aplikace nefungovala správně, kvůli nemožnosti využití některých funkcí zařízení nebo nepokrytí nějakých funkcí určitým frameworkem pro multiplatformní vývoj. Tím má vývojář úplnou kontrolu nad aplikací a může s ní prakticky jakkoliv využívat vše, co daná platforma nabízí.

1.3.2 Nevýhody

Jak již bylo zmíněno výše, nativní vývoj je dražší záležitostí, a to v případě požadavku pro pokrytí více platforem. Každá platforma podporuje jiné jazyky, ve kterých jsou aplikace psány a tím je i potřeba na každou platformu určit vývojový tým zvlášť. A to se týká jak front-endu, tak i back-endu. To, co je naprogramováno pro Android platformu tedy nejde použít prakticky pro iOS platformu a naopak. Co se týče aktualizací, nastává stejný problém.

Pokud je potřeba aplikaci aktualizovat, je nutné aktualizaci navrhnout na každou platformu zvlášť. Kvůli tomu je potřeba mít rozdílné dovednosti a znalosti pro vývoj a údržbu stejné aplikace na rozdílných platformách. Pokud tedy není ve firmě více týmů, je celkový vývoj aplikace časově náročnější.

1.3.3 Android Studio

Android Studio je vývojové prostředí, založené na IntelliJ IDEA, které slouží výhradně pro vývoj aplikací se zacílením, jak již název napovídá, na Android platformu. Za jeho vznikem a vývojem stojí firma Google a jedná se o oficiální vývojové prostředí pro vývoj Android aplikací. Prostředí podporuje vývoj jak v Javě, tak i v Kotlinu. Pro testování aplikace využívá emulátory a lze testovat na jakémkoliv Android zařízení z nabídky a upravovat si nastavení jako je velikost RAM, přední/zadní kamera nebo třeba přítomnost SD karty. Tudíž lze vyzkoušet různé funkce aplikace bez fyzického zařízení. Pro verzi operačního systému je potřeba si stáhnout a nainstalovat příslušné SDK. To vše je stále zahrnuto v IDE a vše se dá nastavit přímo v něm. Samozřejmě lze aplikaci testovat rovnou na fyzickém zařízení, které vývojář připojí přes USB kabel k počítači a aplikaci pak testuje přímo na tomto zařízení. Pro design aplikace je možné použít techniku Drag-and-Drop kdy potřebné prvky přetáhne do výsledného GUI a nemusí parametry (velikost, pozici atd.) psát ručně. Android Studio je dostupné jak pro zařízení s Windows, tak s MacOS a nebo Linux. Co se týče multiplatformního hybridního vývoje (např. Ionic), je potřeba mít nainstalováno toto prostředí a přes něj se aplikace může spustit a testovat na Android zařízení. Vývojové prostředí lze jednoduše spojit s GitHubem a lze tedy verzovat veškerý svůj postup vývoje aplikace bez softwaru třetích stran. [37]

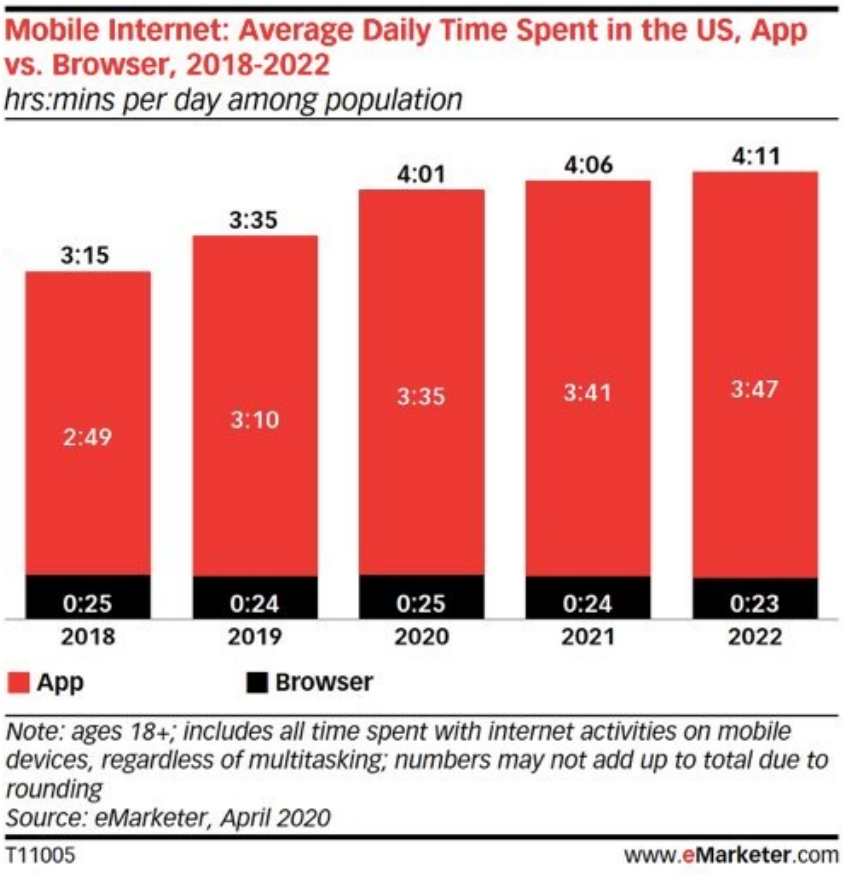
1.3.4 Xcode

Xcode je oficiální nástroj od firmy Apple pro vývoj iOS a MacOS aplikací a je zdarma distribuován přes App Store. Využívá především jazyky Objective-C nebo Swift, ale podporuje i jazyky C, C++ , Objective-C++, Java, AppleScript, Python, Ruby, ResEdit. Xcode je potřeba pro jakýkoliv vývoj aplikaci pro Apple, kdy sice existují i nástroje třetích stran, ty se však nevyhnou problémům jak při vývoji a testování, tak i při finálním nasazení. Proto je Xcode nejvíce používaným prostředím pro vývoj Apple aplikací. Bohužel, Xcode je výhradně Apple aplikace a na zařízeních s Windows operačním systémem jej není možné spustit. Proto je potřeba vlastnit/vypůjčit si zařízení s MacOS nebo využít cloudu, který je ovšem placený. To se samozřejmě týká i multiplatformního hybridního vývoje, kdy je potřeba mít

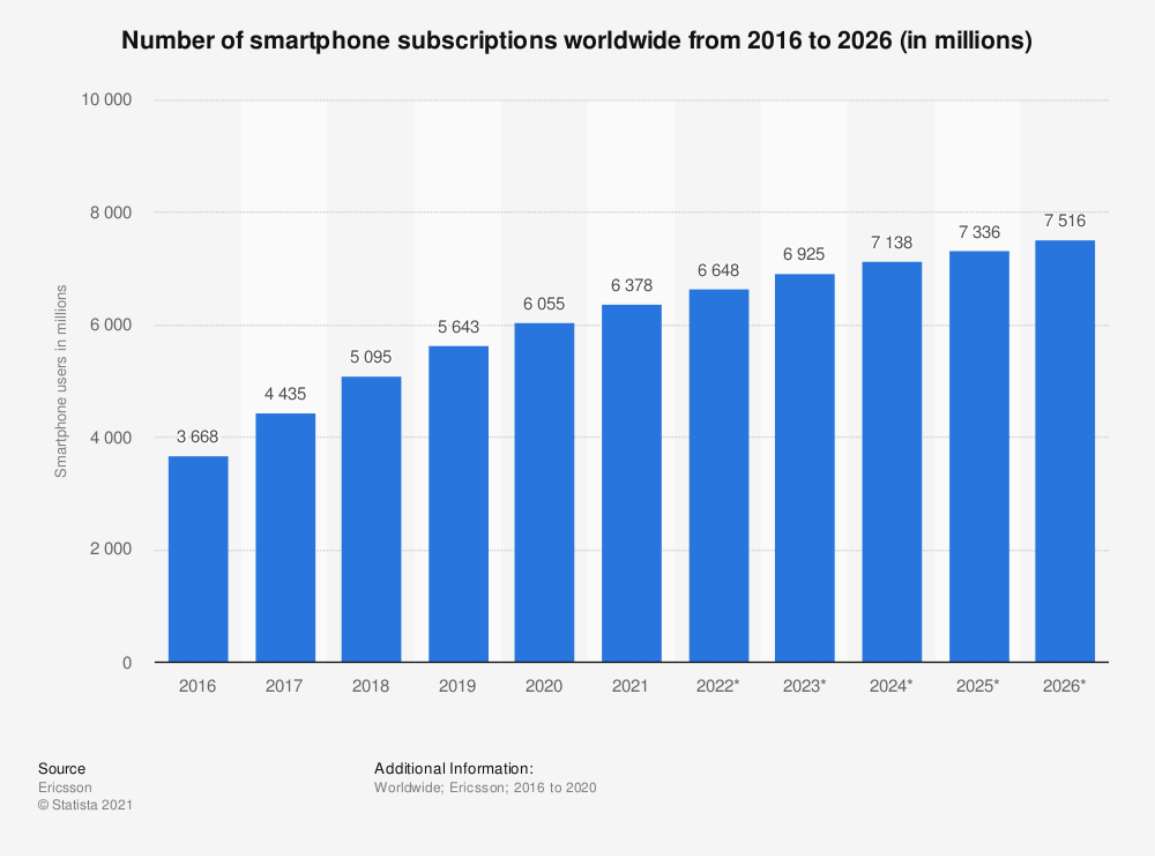
přístup k zařízení s MacOS s Xcodem. Pro programátory, začínající s vývojem pro Apple platformy, nabízí Xcode, místo projektu reálné aplikace, projekt Playground, kde se veškeré změny ihned projeví na simulátoru (popř. fyzickém zařízení) a to ať už v textové nebo grafické formě. Je to výhodné právě pro naučení jazyka Swift a také seznámení s celým vývojovým prostředím Xcode. [18]

1.4 Progresivní webové aplikace

Progresivní webové aplikace vznikly hlavně kvůli velké návštěvnosti webových stránek a také velké rozšířenosti mobilních telefonů. Na Internet se dle průzkumu eMarketeru v USA připojuje více uživatelů z aplikací než z prohlížeče (viz. Obrázek 2) [16]. Webová aplikace tedy přiláká uživatele a aplikace v mobilu zajistí jejich návrat. A jak lze vidět i na průzkumu firmy Ericsson, zprostředkovaném společností Statista, je počet uživatelů chytrých mobilních telefonů již přes 6 miliard (viz. Obrázek 3). [17]



Obrázek 2. Průměrný čas strávený na webu nebo v aplikaci v USA [16]



Obrázek 3. Počet uživatelů chytrých mobilních telefonů na celém světě [17]

Mezi PWA patří webové aplikace psané pomocí HTML, CSS a JavaScriptu,. Jedná se prakticky o dynamickou webovou aplikaci, kterou si uživatel může stáhnout přímo z webu. Ta se pak chová skoro jako nativní aplikace. Obsahuje svůj offline režim, s uživatelem komunikuje pomocí push notifikací (upozornění) a přes API dokáže komunikovat s hardwarem zařízení. Aplikace však funguje i přímo na webu a není tedy nutno ji stahovat do zařízení. Tato technologie díky svým prvkům má asi nejblíže k hybridním aplikacím. [4]

1.4.1 Výhody

Mezi největší výhody bezpochyby patří jednoduchost vývoje a také to, že díky responzivitě se aplikace dá spustit prakticky na každém zařízení, ať už desktopovém nebo mobilním. Tím, že je aplikace distribuována on-line, postačuje uživatelskému zařízení pouze prohlížeč a přístup k Internetu. Díky komunikaci přes HTTPS je aplikace zabezpečena např. při odesílání hesel. Aplikace díky Service Workeru dokáže pracovat off-line, takže po nainstalování funguje i bez připojení k internetu. Protože se jedná o webovou aplikaci, tak je její vzhled na všech zařízeních jednotný a pouze se přizpůsobuje velikosti obrazovky. Progresivní webové

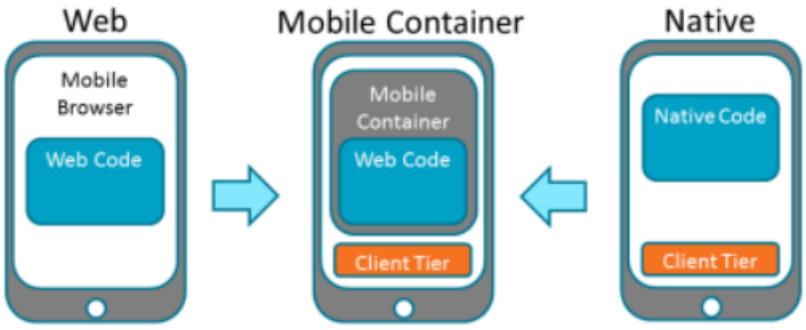
aplikace jsou vždy aktuální a při úpravách na straně provozovatele se veškeré změny projeví i u uživatelů ihned po připojení k Internetu. Další výhodou je i jednoduchá propagace, protože uživatel může být přímo přesměrován na web a aplikaci si stáhnout. [5] [6]

1.4.2 Nevýhody

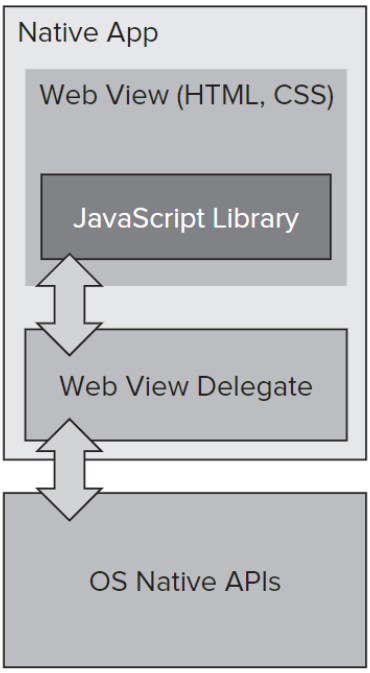
Nevýhodou PWA je větší spotřeba baterie, a i přes schopnost fungování v off-line režimu a přístupu k hardwaru, aplikace nenahradí plnohodnotně nativní aplikaci nebo aplikaci s nativními prvky. [6] Progresivní webové aplikace také nejsou podporovány u starších prohlížečů, na rozdíl od moderních, kde mají větší podporu.

1.5 Hybridní aplikace

Hybridní aplikace jsou jednoduše hybrid mezi webovou a nativní aplikací. Aplikace je napsána pomocí webové technologie za pomoci HTML, CSS a JavaScriptu a jsou to v podstatě webové aplikace, které byly vloženy do nativního kontejneru nebo používají WebView objekt (grafické znázornění viz. Obrázek 4 a Obrázek 5). Tento objekt při každém použití aplikace zobrazí obsah díky webovým technologiím (HTML, CSS, JavaScript). Jakmile je aplikace stažena z App Store a lokálně nainstalována, tak se může připojit k jakýmkoli možnostem zařízení, které mobilní platforma poskytuje. To vše prostřednictvím prohlížeče, který je součástí aplikace. Díky tomu má aplikace přístup i k webovým službám. Zabudovaný prohlížeč a jeho zásuvné moduly běží na back-endu a jsou pro koncového uživatele prakticky neviditelné. [7] [8] [21] [22] [24]



Obrázek 4. Grafické znázornění nativního kontejneru [21]



Obrázek 5. Grafické znázornění aplikace používající Web View [20]

Pokud vývojář uvažuje nad volbou mezi PWA a hybridní aplikací, musí zvážit, zda je potřeba přístup právě k nativním funkcím. Mobilní webové aplikace sice mají jednodušší upgrade a zavádění aktualizací, ovšem některé funkce dostupné pouze pro nativní aplikace zařízení nemohou být dostupné pro webové aplikace založené na prohlížeči. Pokud například aplikace potřebuje složitý bezpečnostní proces, který nemohou zpracovat webové aplikace, například přístup ke snímači otisků prstů, snímači čárových kódů nebo jinému zařízení, neexistuje způsob, jak toho dosáhnout ze standardního prohlížeče. Hybridní aplikace lze rozšířit tak, aby poskytovaly tuto funkcionalitu, zatímco čistě webové aplikace nikoli. [20]

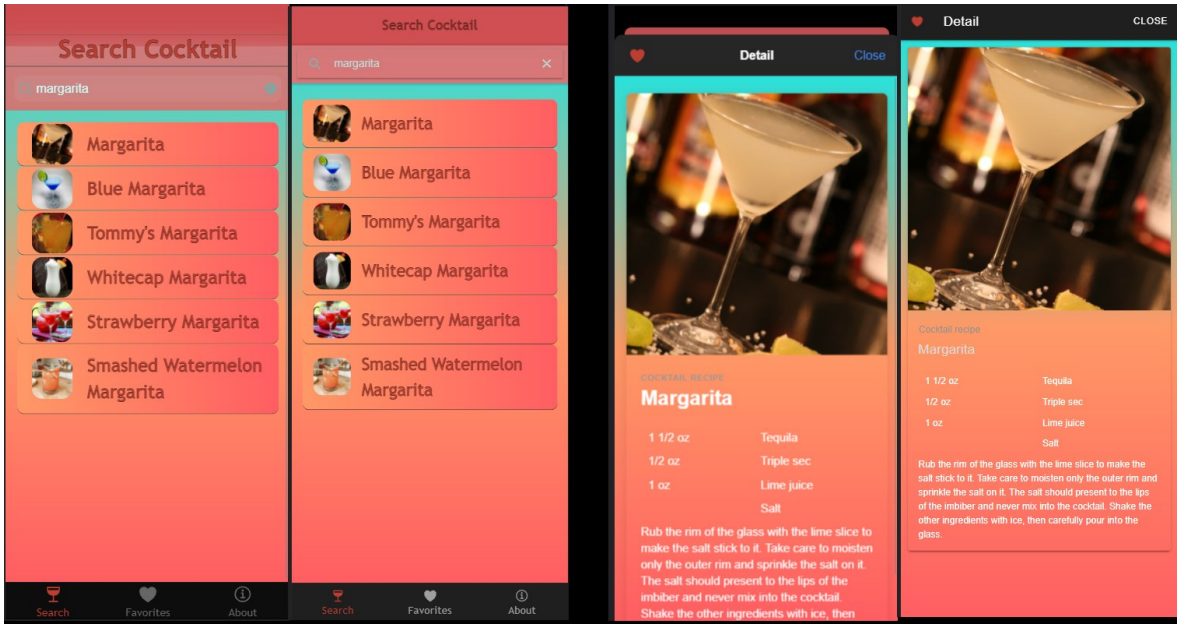
Mezi nejznámější nástroje pro vývoj hybridních aplikací patří Ionic Framework a Electron.

1.5.1 Výhody

Aplikace je schopna fungovat na různých platformách, což je i cílem této technologie. Stále je rychlejší vývoj a sestavení aplikace ve srovnání s nativními aplikacemi a tím je i vývoj levnější než vytváření dvou verzí nativní aplikace pro dvě různé platformy. Tím, že je základ aplikace jednotný, je snazší spouštění oprav a aktualizací. Oproti čistě webovým aplikacím, můžou hybridní aplikace pracovat online i offline (pokud ovšem není potřeba komunikovat s databází, zde offline režim postrádá smysl). [7] [8]

1.5.2 Nevýhody

Mezi zásadní nevýhodu patří rozdíly způsobené nakloněním vývoje na jednu platformu – například pokud vývojový tým opře svou práci o jednu platformu, jiná podporovaná platforma může postrádat kvalitu nebo trpět chybami (viz. Obrázek 6). Tím se může vzhled aplikace lišit platformu od platformy nebo některé funkce nemusí fungovat tak, jak by měly. Proto je potřeba otestovat aplikaci na řadě zařízení pro zajištění správného fungování. Pokud by tato skutečnost byla vývojovým týmem ignorována, uživatelská zkušenost (UX) se může zhoršit. To platí i pokud uživatelské rozhraní (UI) není podobné a dostatečně dobře navržené pro prohlížeče, na které je uživatel zvyklý. Proto by měli vývojáři přemýšlet nad UX a UI designem pro každou platformu zvlášť. [7] [8] Další nevýhodou je určitě potřeba nativních pluginů pro použití funkcí zařízení, jako je například Touch ID nebo fotoaparát. Pokud plugin neexistuje, není tedy, bohužel, možné tuto funkci v aplikaci použít. Také jedna z nevýhod, hlavně u složitějších aplikací, je neúplné pokrytí vývoje pouze multiplatformním nástrojem a je potřeba nějakou část vyvinout nativně. Tudíž se vývojový tým na 100% nevyhne nativnímu kódu.



Obrázek 6. Rozdíl ve vzhledu Ionic aplikace na dvou platformách (iOS|Android)

1.5.3 Ionic Framework

Ionic Framework (dále jen Ionic) je open-source nástroj pro vývoj jak hybridních, tak webových aplikací (PWA). Pro přístup k nativním funkcím zařízení (fotoaparát, poloha, Bluetooth atd.) používá nativní pluginy a pomocí JavaScriptu s nimi dále pracuje. Pokud

neexistují pluginy na nějaký problém, je možné přejít do plně nativního SDK. Pro front-end má Ionic integrovanou podporu pro frameworky Angular, React, Vue nebo bez frameworku s čistým vanilla JavaScriptem. Díky práci s webovými technologiemi jsou komponenty uživatelského rozhraní v Ionicu moderní s vysokou kvalitou. Ionic nabízí také spoustu pluginů pro různé funkce aplikace. Také je Ionicem podporováno rozšíření od třetích stran. Některá jsou oficiální a placená (např. Apple Pay, Google Pay, SQLite…) a některá komunitní (např. Google Maps, Twitter, Facebook). Pro nativní části aplikace je potřeba do projektu přidat zásuvné moduly buď Capacitoru (novější) nebo Cordovy (zastaralejší). V dnešní době je lepší používat spíše Capacitor, protože je i ve velké míře kompatibilní s Cordovou. [9]

1.5.4 Electron

Electron je javascriptový framework pro tvorbu desktopových aplikací. Pro počítače s operačním systémem Windows, macOS nebo Linux tedy stačí napsat aplikaci pomocí Electronu pouze jednou, a tím pokrýt všechny tyto platformy. Stejně jako Ionic používá k tvorbě především webové technologie (HTML, JavaScript a CSS). Jedná se o open-source projekt pod záštitou GitHubu, ke kterému ale přispívají i jiné firmy, jako například Microsoft nebo Slack. Electron pro práci se systémem zařízení používá NodeJS a pro vykreslování používá Chromium. Tím jsou pokryté funkce jak prohlížeče, tak funkce Node ekosystému. Největší nevýhodou pak je nejspíše ta, kdy i pro malé aplikace je potřeba startovat celý prohlížečový engin, který využívá dost velkou část paměti. Kvůli tomu je pak i výsledný balíček poměrně velký, kdy obyčejná Hello World aplikace může mít více než 30 MB, a zvláště pokud aplikace pracuje s větším množstvím dat, může mít problémy s výkonem. Je proto i dobré dávat si pozor na množství vykreslovaných prvků. GitHub pravidelně vydává optimalizace jádra a tím se snaží zrychlit aplikace psané pomocí Electronu. Za pomoci tohoto frameworku jsou vytvořeny známé aplikace jako VS Code, Microsoft Teams, Whatsapp nebo třeba Twitch. [10] [11] [12]

1.6 Multiplatformní aplikace

Vývoj aplikací napříč platformami sdílí stejnou kódovou základnu jako hybridní vývoj. Mobilní vývojáři používají k vytvoření aplikace stejné nativní prvky, ale liší se v poskytování lepší uživatelské zkušenosti s různým uživatelským rozhraním. Ve srovnání s již zmíněnými hybridními aplikacemi, tyto technologie využívají různé jazyky pro vývoj back-end části a

mají lepší přístup k funkcím zařízení. Výkon uživatelského rozhraní se shoduje s nativními aplikacemi, protože sdílí podobnou kódovou základnu. [19] [22]

Přesto se velmi často stává, že se tyto technologie míchají a není jednoznačné, která je čistě hybridní a která je spíše multiplatformní. Mají spoustu společného a využití na trhu je také velmi podobné. Dokonce i průzkumy zahrnují oba druhy technologií (viz. Obrázek 1). Pro uživatelské a grafické rozhraní používají většinou jiné technologie než hybridní vývoj, který je převážně tvořen webovými technologiemi.

1.6.1 Výhody

Multiplatformní vývoj skýtá mnoho výhod jak už pro programátora, tak pro zákazníka. Dokáže ušetřit spoustu financí i času na vývoji a vyžaduje menší soubor znalostí vývojáře [13]. Pokud vývojář zvolí správný framework pro tvorbu aplikace, je možné dosáhnout až 90-100% znovupoužitelnosti kódu pro všechny vyvíjené platformy s minimalizací nativních částí. Oproti hybridním aplikacím lépe spolupracují s funkcemi zařízení a mají s nimi menší komplikace.

1.6.2 Nevýhody

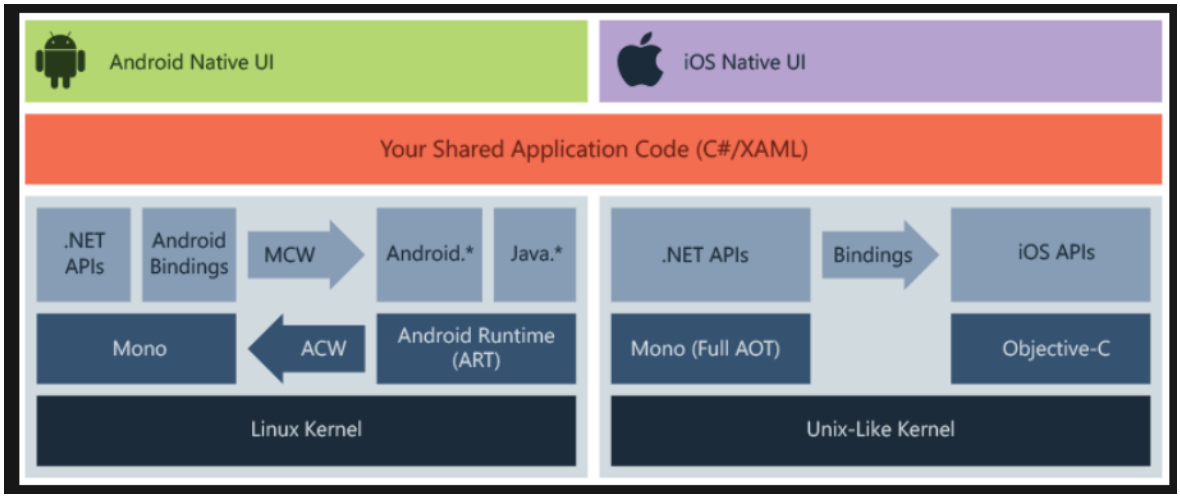
Jedna z nevýhod, kterou si musí programátor ujasnit ještě před začátkem vývoje je ta, že multiplatformní aplikace není zárukou rozběhnutí dané aplikace na všech platformách. Každá technologie je vždy zaměřená na konkrétní platformy. S tím pak musí podle daného problému každý vývojář počítat a tomu přizpůsobit postup a způsob vývoje. Například nástroj Xamarin od Microsoftu byl zaměřen na Android i iOS, jeho budoucí nástupce .NET MAUI je však zaměřen už i na desktopová zařízení s Windows, macOS i Linux [13]

Další nevýhodou je velikost výsledného balíčku. Například aplikace v Xamarinu obsahuje všechny knihovny.

1.6.3 Xamarin

Xamarin je open-source platforma pro vytváření moderních a výkonných aplikací pro iOS, Android a Windows s .NET, kde slouží jako abstraktní vrstva, která spravuje komunikaci sdíleného kódu s kódem základní platformy. Běží ve spravovaném prostředí, které poskytuje vymoženosti, jako je alokace paměti a garbage collection. Xamarin umožňuje vývojářům sdílet v průměru až 90 % jejich aplikací napříč platformami. Tento vzor umožňuje vývojářům psát veškerou svou business logiku v jediném jazyce (nebo opakovaně používat

existující kód aplikace), ale na rozdíl od hybridních aplikací mohou dosáhnout nativního výkonu, vzhledu a chování na každé platformě. Aplikace Xamarin mohou být napsány na PC nebo Mac a zkompilovány do nativních aplikačních balíčků, jako je soubor .apk na Androidu nebo soubor .ipa na iOS. Aplikace se vytváří v jazyce C# a XML. Na obrázku níže (Obrázek 7) lze vidět architektura Xamarin aplikace. [23]



Obrázek 7. Architektura Xamarin aplikace [23]

1.6.4 React Native

React Native je JavaScriptový framework pro psaní nativně vykreslovaných mobilních aplikací pro iOS a Android. Je založen na Reactu, JavaScriptové knihovně, od firmy Facebook, pro tvorbu uživatelských rozhraní, ale místo cílení na prohlížeč cílí na mobilní platformy. Tedy vývojáři, kteří tvoří weby v Reactu, mohou díky React Nativu psát mobilní aplikace, které vypadají a chovají se jako nativní, a to vše za pomoci JavaSriptové knihovny, kterou již znají. React Native je tedy napsán pomocí směsi JavaScriptu a značkovacího jazyka XML, známého jako JSX. React Native poté vyvolá nativní API pro vykreslování v Objective-C (pro iOS) nebo Javě (pro Android). Tím zařídí vykreslování pomocí skutečných komponent mobilního uživatelského rozhraní, nikoli pomocí webview. Aplikace pak vypadá a působí jako jakákoli nativní aplikace. Díky přístupu k nativní API hostitelské platformy může aplikace přistupovat k funkcím platformy jako je fotoaparát telefonu nebo poloha uživatele. Další výhodou přímé komunikace s API platformy, na rozdíl od hybridních technologií, které využívají webview, jsou různé animace a vykreslování grafických prvků hladší a rychlejší. Aplikace je tím výkonnější a mnohem rychleji reaguje na změny. [24]

1.6.5 Flutter

Flutter je open-source framework od firmy Google pro multiplatformní vývoj mobilních aplikací. Umožňuje tedy vytvořit nativní mobilní aplikaci pouze s jednou kódovou základnou. Skládá se ze dvou částí:

- SDK (Software Development Kit): Sada nástrojů, které mají pomoci s vyvíjením aplikace. To zahrnuje nástroje pro kompilaci kódu aplikace do nativního strojového kódu (kód pro iOS a Android).
- Framework (UI knihovna založená na widgetech): jedná se o kolekci opakovaně použitelných prvků UI (tlačítka, textové vstupy, posuvníky atd.), které je možné přizpůsobit podle potřeby vývojáře či klienta.

Flutter využívá jazyk Dart, také od firmy Google, který je podobný Javascriptu. Zaměřuje se především na vývoj front-endu a může být použit k vytváření mobilních i webových aplikací. [25]

2 BEZPEČNOST A ZABEZPEČENÍ MOBILNÍCH APLIKACÍ

Mobilní aplikace, tak jako každá jiná, musí splňovat bezpečnostní požadavky, které se týkají jak soukromých dat uživatelů, tak jejich ochraně před zneužitím jejich osobního majetku. Proto by každá mobilní aplikace měla mít přístup jen k datům a funkcím, které potřebuje ke svému správnému chování, jenž uživatel očekává, a k těm, k nimž dá uživatel povolení. Aplikace nesmí používat citlivá data a zároveň by k nim neměla mít ani přístup, aniž by jí to uživatel schválil. Pokud aplikace potřebuje komunikovat s nějakým systémem mimo zařízení a tato komunikace může být pro uživatele citlivá, musí být tato komunikace taktéž zabezpečena různými metodami.

2.1 Oprávnění přístupu k systémovým funkcím

Oprávnění aplikace pomáhají podporovat soukromí uživatelů tím, že chrání přístup k následujícímu:

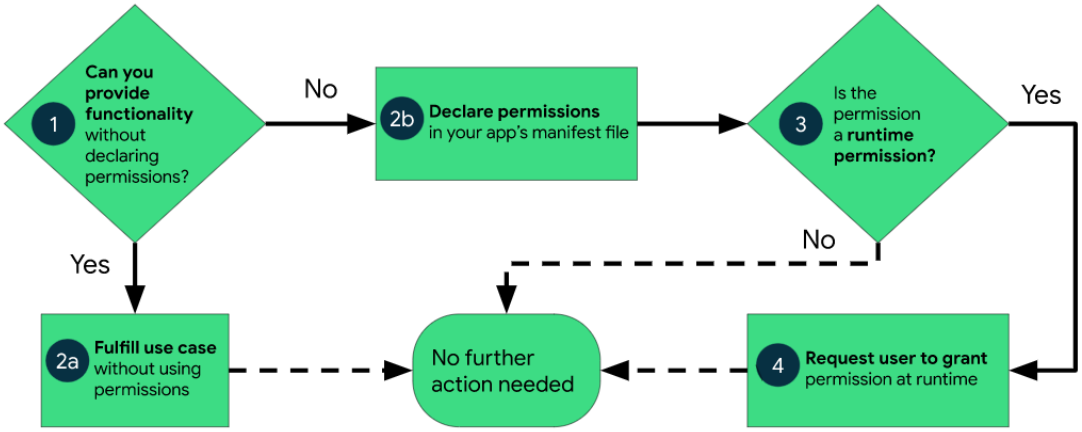
- Omezená data, jako je stav systému a kontaktní informace uživatele.
- Omezené akce, jako je připojení ke spárovanému zařízení a nahrávání zvuku.

Každá aplikace by měla být při využívání těchto dat a akcí naprosto transparentní a uživatel, který si aplikaci plánuje stáhnout a nainstalovat, by měl znát všechna oprávnění, které by aplikace po něm mohla chtít. Tato oprávnění musí odpovídat účelu aplikace, obzvláště pokud se jedná o přístup k nebezpečným datům nebo akcím. Tyto detaily tedy aplikace musí předat uživateli, aby byla důvěryhodná a uživatel jí mohl věřit, že není nijak nebezpečná. [26]

Pokud aplikace nabízí funkce, které mohou vyžadovat přístup k omezeným datům nebo omezeným akcím, je potřeba určit, zda je možné získat informace nebo provést akce, aniž by musela být deklarována oprávnění. V aplikaci je možné využít různé funkce a pokrýt případy použití (např. fotografování, pozastavení přehrávání médií nebo zobrazování relevantních reklam), a to bez potřeby deklarace jakýchkoliv oprávnění.

Pokud je nezbytné, aby aplikace potřebovala přistupovat k omezeným datům nebo provádět omezené akce pro plnění své správné funkce, je nutné deklarovat příslušná oprávnění. Některá oprávnění, známá jako oprávnění při instalaci (install-time permissions), jsou automaticky udělena při instalaci aplikace. Pro oprávnění při využívání aplikace, známá jako oprávnění za běhu (run-time permissions), je potřeba, aby aplikace požádala o oprávnění za

běhu aplikace. Následující obrázek ilustruje pracovní postup pro používání oprávnění aplikace na platformě Android (Obrázek 8).

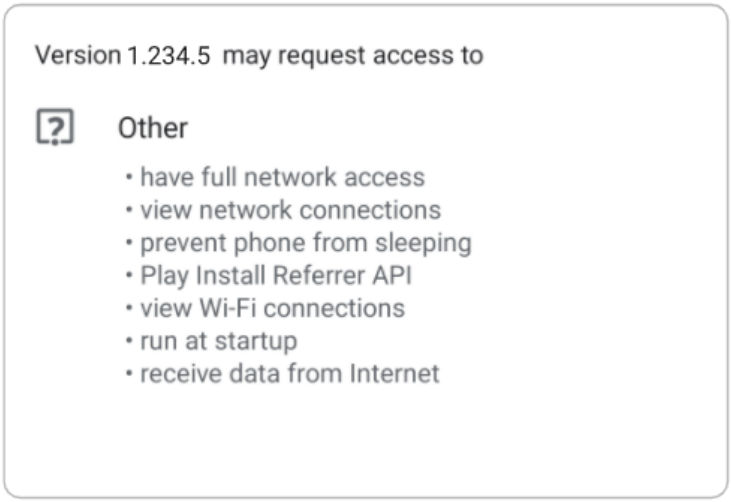


Obrázek 8. Pracovní postup pro používání oprávnění v systému Android [3]

Android tedy kategorizuje oprávnění do různých typů, včetně oprávnění při instalaci, oprávnění za běhu a speciálních oprávnění. Každý typ oprávnění označuje rozsah omezených dat, ke kterým má aplikace přístup, a rozsah omezených akcí, které může aplikace provádět, když systém dané aplikaci udělí potřebná oprávnění. [3]

2.1.1 Install-time permissions

Oprávnění při instalaci dávají aplikaci omezený přístup k omezeným datům a umožňují jí provádět omezené akce, které minimálně ovlivňují systém nebo jiné aplikace. Pokud jsou v aplikaci deklarována tato oprávnění, systém je pak aplikaci automaticky udělí při její instalaci. Obchod s aplikacemi zobrazuje uživateli, při zobrazení stránky s podrobnostmi o aplikaci, oznámení o povolení instalace (viz. Obrázek 9).



Obrázek 9. Výpis install-time oprávnění v obchodu s aplikacemi [3]

Android obsahuje několik podtypů install-time oprávnění, včetně normálních oprávnění (normal permissions) a oprávnění po podepsání (signature permissions). [3]

2.1.1.1 Normal permissions

Tato oprávnění umožňují přístup k datům a akcím, které přesahují aplikační sandbox. Data a akce však představují velmi malé riziko pro soukromí uživatele a provoz jiných aplikací. [3]

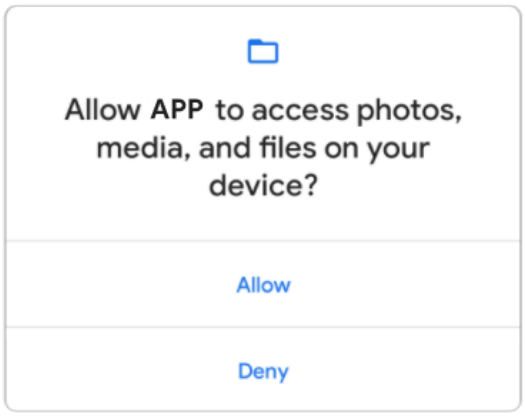
2.1.1.2 Signature permissions

Pokud aplikace deklaruje signature permissions, které definovala jiná aplikace, a pokud jsou obě aplikace podepsány stejným certifikátem, systém udělí oprávnění první aplikaci v době její instalace. V opačném případě nelze této první aplikaci udělit oprávnění. Zároveň některá signature permissions nejsou určena pro aplikace třetích stran. [3]

2.1.2 Runtime permissions

Oprávnění za běhu, známá také jako nebezpečná oprávnění, poskytují aplikaci přístup k omezeným datům a umožňují jí provádět omezené akce, které podstatněji ovlivňují systém a další aplikace. Než však aplikace bude mít přístup k omezeným datům nebo provádět omezené akce, musí nejdříve požádat o oprávnění za běhu. Pokud aplikace požaduje oprávnění za běhu, systém zobrazí výzvu k povolení provedení dané akce nebo k přístupu k datům (viz. Obrázek 10). Mnoho runtime oprávnění přistupuje k soukromým uživatelským datům, což

je zvláštní typ omezených dat, která obsahují potenciálně citlivé informace. Příklady soukromých uživatelských dat zahrnují informace o poloze a kontaktních údajích. [3] [26]



Obrázek 10. Výzva k povolení k přístupu k omezeným datům/funkcím [3]

2.1.3 Special permissions

Speciální oprávnění odpovídají konkrétním operacím aplikace. Speciální oprávnění mohou definovat pouze platforma a výrobci OEM. Platforma a výrobci OEM navíc obvykle definují speciální oprávnění, když chtějí chránit přístup k obzvláště silným akcím, jako je například kreslení přes jiné aplikace. Stránka **Special app access** v nastavení systému obsahuje sadu uživatelsky přepínatelných operací. Mnoho z těchto operací je implementováno právě jako speciální oprávnění.

2.2 Útoky na mobilní aplikace a jejich předcházení

Díky vztahu, jaký v současné době má většina populace k chytrým mobilním telefonům a aplikacím, které nabízí, je po prolomení slabších míst aplikace jednoduché získat soukromá data uživatele zařízení. Jediným „vloupáním“ by mohl útočník znát uživatelovo jméno, věk, domácí adresu, čísla účtů a dokonce i aktuální polohu s přesností na několik metrů. Pro aplikace v institucích a firmách to platí dvojnásobně. Podnikové aplikace si totiž vyměňují mimořádně citlivé informace jak už firemní, tak osobní (data řídících osob i zaměstnanců). Na firemní data se zaměřují nejvíce, obzvláště pokud se jedná o finanční instituci. S tímto druhem informací v sázce musí vývojáři mobilních aplikací udělat vše, co mohou, aby ochránili své uživatele a klienty. Pokud by se nějaké prolomení aplikace stalo, vývojář aplikace tak může poškodit svou reputaci a důvěryhodnost a odradit stálé i potencionální klienty. [28] [29]

2.2.1 Bezpečný kód

Chyby a zranitelnosti v kódu jsou výchozím bodem, který většina útočníků používá k proniknutí do aplikace. Pomocí reverzní analýzy kódu se pokusí najít tyto chyby a dále s ním manipulovat. Vývojář by měl dbát, aby při kódování aplikace používal zabezpečený framework, a tím se vyhnul chybám v kódu programu. Jakékoli mezery v kódu a designu mohou útočníkům poskytnout přístup k citlivým a osobním uživatelským informacím. To jisté platí i o knihovnách a frameworcích třetích stran. Pokud je framework nezabezpečený, nebo má v sobě chybu, vývojář není schopen tuto chybu najít a opravit. Proto by měl přemýšlet nad nástroji, které potřebuje použít ve své aplikaci, aby jejich bezpečnost nenarušila bezpečnost jeho výtvoru. [29]

Pro ochranu před takovým útokem je možné minifikovat a poměnit kód, aby jej nebylo možné zpětně analyzovat a jednoduše číst. Dále je potřeba periodicky testovat a opravovat případné chyby, jakmile se objeví. Tomu napomůže návrh kódu tak, aby jej bylo možné snadno aktualizovat a opravovat. Navíc je možné ochránit aplikaci pomocí RASP (Runtime Application Self-Protection) a tím zabránit a odhalit kybernetické útoky v reálném čase. Při vývoji a údržbě je také možnost si najmout třetí stranu, aby se pokusila na aplikaci zaútočit a otestovala, jak bude aplikace reagovat na možné útoky. [28] [29]

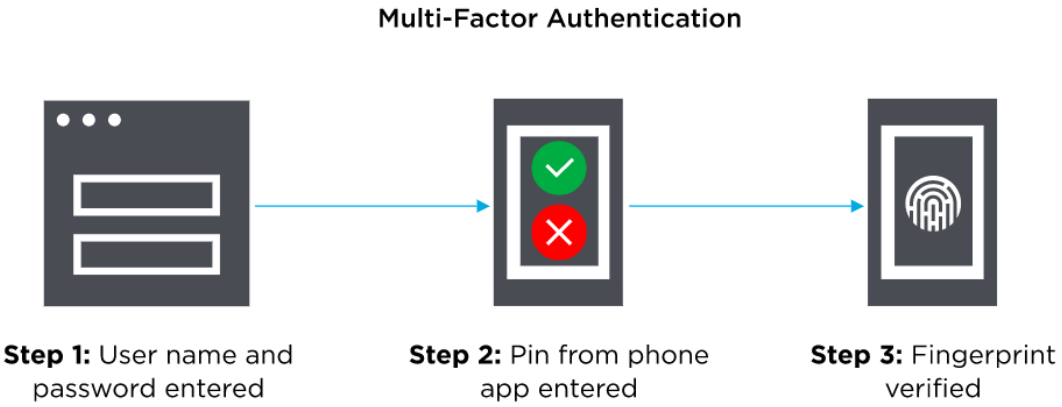
2.2.2 Šifrovaná komunikace

Každá komunikace aplikace s okolím musí být šifrována. To znamená, že veškerá data by například při útoku Man-in-the-middle byla pro útočníka neznámá, protože by neměl potřebný klíč k dešifrování. Jediné, co by takto získal, je pouze změť znaků, které nemusejí mít ani žádný opakovaný vzor. Co útočník získá záleží také na šifrování, jenž je v aplikaci použito, a to se odvíjí od šifrovaných dat. [29]

2.2.3 Zesílení autentizace a identifikace uživatele

Pro ochranu a přístup do aplikace a dat s ní spojenými slouží autentizace a identifikace uživatele. K tomu slouží přihlašovací jméno a heslo. Aby uživatel ztížil útok, je lepší, když si nastaví silné heslo s písmeny, čísly a znaky a nejlépe náhodně. Pro posílení této bezpečnostní funkce se dá použít MFA (Multi-Factor Authentication) nebo 2FA (Twofactor Authentication). MFA je například zadání pinu, nebo ověření otiskem prstu (popř. ověření obličejem) hned po zadání přihlašovacích údajů (viz. Obrázek 11). 2FA je pak použití kódu, který aplikace pošle uživateli na e-mail, nebo z autentizační aplikace třetích stran. MFA se často

zaměňuje s dvoufaktorovou autentizací (2FA). 2FA je v podstatě podmnožinou MFA, protože 2FA omezuje počet faktorů, které jsou vyžadovány, pouze na dva faktory, zatímco MFA mohou být dva nebo více. [27] [28] [29]



Obrázek 11 . Vícefaktorové ověřování [27]

2.2.4 Bezpečná API

API, která nejsou autorizována a jsou volně programována, mohou neúmyslně udělit útočníkovi oprávnění, která mohou být zneužita. Například místní ukládání informací o autorizaci do mezipaměti sice pomáhá programátorům snadno znovu použít tyto informace při volání API a celkové používání rozhraní API, ale poskytuje také mezeru, jejímž prostřednictvím mohou zneužít oprávnění. Proto je doporučeno, aby byla API pro maximální bezpečnost autorizována centrálně. [29]

Zabezpečení mobilních aplikací tedy závisí na zabezpečených API. Bezpečnost rozhraní API lze maximalizovat použitím různých strategií: [28]

- Skryjte všechny klíče zabezpečení API
- Autentizace a autorizace prostřednictvím PBAC (řízení přístupu založeného na zásadách), RBAC (řízení přístupu založeného na rolích) nebo CBAC (řízení přístupu založeného na obsahu)
- Implementace šifrování
- Implementace řádné validace
- Použití auditování a logování
- Použití rozhraní API RESTful (přenos reprezentativního stavu)
- Použití zdrojových kvót a omezení

2.3 SQL Injection

SQL Injection je jeden z nejčastějších útoků na aplikace využívající databázi. Jedná se prakticky o injektování, neboli vkládání nežádoucího a mnohdy i škodlivého dotazu do databáze, nad kterou aplikace pracuje. Útočník může získat veškeré informace o databázi a jejích tabulkách a tím i získat veškerá data z těchto tabulek. Také může data i upravovat nebo mazat, a tím poškodit celou databázi. To vše zvládne přes neošetřené vstupy, kdy aplikace tyto vstupy předává přímo do dotazu tak, jak jí jsou předány, bez jakýchkoli validací, a tím spustit příkaz, který aplikace neměla v úmyslu spustit. Do url webové aplikace by mohl útočník zadat například [*https://e-shop.cz/products?category=Slevy' UNION SELECT username, password FROM users--*] a tím vytáhnout uživatelské jméno a heslo z tabulky *users*. [30]

2.3.1 Ochrana před SQL Injection

Nejlepší ochranou proti tomuto typu útoku je použití parametrizované dotazy (Prepared Statements) při skládání SQL dotazů. Parametrizované dotazy si programátor může vytvořit sám, nebo pro usnadnění a větší bezpečnost existuje spoustu knihoven podle typu aplikace a technologie, pomocí které je aplikace tvořena, které generují již parametrizované dotazy. Parametrizace zaručí, že se s parametry SQL dotazu zachází bezpečným způsobem. Zastaralejším a nebezpečným typem ochrany je tzv. „escapování“, kdy se ve vstupu potencionálně škodlivé znaky upraví na součást řetězce. To ovšem nefunguje na příkazy, kde se tyto znaky nepoužívají a proto je tato metoda nebezpečná. Dále je důležitá ochrana na straně databáze, kde by mělo být správně nastaveno oprávnění uživatele. Kdyby se tedy podařilo útočníkovi odeslat jeho vlastní SQL dotaz, databáze by mu měla zabránit v přístupu k tabulkám nebo jejich úpravě či mazání. [31]

Dále se pro ochranu používá objektově-relační mapování, neboli ORM, které poskytuje objektově orientovanou vrstvu mezi relačními databázemi a objektově orientovanými programovacími jazyky bez nutnosti psát SQL dotazy. ORM data z OOP datových objektů přeloží a vytvoří strukturovanou mapu, která vývojářům pomáhá porozumět základní struktuře databáze. Mapování představuje jak objekty souvisí s různými tabulkami. ORM používají tyto informace k převodu dat mezi tabulkami a generování kódu SQL pro relační databázi k vkládání, aktualizaci, vytváření a odstraňování dat v reakci na změny, které aplikace provede v datovém objektu. Po použití mapování bude ORM, spolu s dodatečnými knihovnami, spravovat datové potřeby aplikace a nebudete tak potřeba psát příkazy manuálně. [32]

3 ARCHITEKTURA APLIKACE

3.1 Architektura MVVM

Aplikace z většiny používá architekturu MVVM, neboli Model-View-ViewModel.

3.1.1 Model

Model představuje data, se kterými aplikace pracuje. Mohou představovat data z databáze nebo pouze obsahovat nějaké hodnoty, které aplikace používá na prezentování uživateli. V MVVM architektuře jsou většinou Modely obaleny ve ViewModelu, který s nimi pracuje a předává jejich hodnoty příslušnému View. [35]

3.1.2 View

View je představuje GUI a prezentuje uživateli data. Jsou ovládací prvky, viditelné a interaktivní prvky na stránce. Jde tedy o základní elementy, jako jsou tlačítka, tagy a textová pole, až po pokročilejší zobrazení, jako jsou seznamy a navigace. Views obsahují properties, které určují jejich obsah, písmo, barvu a zarovnání. [35]

3.1.3 ViewModel

ViewModel je třída vytvořená pro poskytování dat na konkrétnímu View pomocí jednoho nebo více Modelů, které v sobě zaobaluje. View by totiž nemělo přímo přistupovat k Modelu. ViewModel je opatřen view-specific vlastnostmi a je svázán s View. Jedná se tedy o datový model s tím rozdílem, že je přizpůsoben konkrétnímu View pomocí pomocných tříd a obsluhy událostí, na kterém se naplňují data nebo spravují změny datového Modelu. [35]

II. PRAKTICKÁ ČÁST

4 POPIS APLIKACE *REALTYS* A .NET MAUI

Aplikace byla nazvána *Realtys* a slouží ke správě nemovitostí pro malé investory. Technologie, ve které byla aplikace vytvořena je .NET MAUI.

4.1 .NET MAUI

Pro aplikaci *Realtys* byl vybrán nástroj od Microsoftu .NET MAUI. Jedná se o technologii pro vývoj multiplatformních aplikací pro mobilní zařízení i desktop za použití jazyků C# a XAML. Framework také nabízí možnost s BlazorWebView a tím použít webové technologie To znamená, že jedna aplikace se dá rovnou použít jak na Android a iOS zařízeních, tak i na Windows a macOS. Framework .NET MAUI je nový a dostupný pouze v preview verzi Visual Studio 2022 (Version 17.2.0 Preview 1.0).

4.2 Realtys

Aplikace *Realtys* je nástroj pro správu menšího počtu nemovitostí (v jednotkách až nižších desítkách), tudíž je spíše určena pro menší nebo začínající investory nebo investory, jenž si do aplikace ukládají krátkodobé záznamy. Jedná se o aplikaci, kterou investor bude moci použít například po prohlídce potenciální nemovitosti, kde si zadá potřebné hodnoty, jako je možná výše nájemného nebo náklady spojené se správou nemovitosti. Aplikace mu pak nemovitost uloží a umožní mu pak i zpětnou editaci. Aplikace mu také ukáže roční návratnost a další užitečné hodnoty. Pro výpočet některých hodnot jsou využity vzorce z blogu a podcastu AdolMonitor, který se zaměřuje na investice do nemovitostí [34].

4.2.1 Vzorce použité při výpočtech detailních hodnot

Zde jsou popsány vzorce, které jsou použity při zobrazení detailních hodnot a při ukládání některých hodnot, souvisejících s úvěrem:

- Počáteční dluh:

  $$Cena\ nemovitosti * \frac{podíl\ úvěru\ na\ ceně\ nemovitosti}{100}\ [Kč]$$

- Hodnota měsíční splátky úvěru:

  $$měs.\ úroková\ míra = \frac{roční\ úroková\ míra}{12 * 100}$$

  $$v = \frac{1}{1 + měs.\ úroková\ míra}$$

$$měs.\,splátka\ úvěru = \frac{měs.\,úroková\ míra * počáteční\ dluh}{1 - v^{počet\ měsíců\ splácení\ úvěru}}\ [Kč]$$

- Střední hodnota splátky úroku a jistiny (úmoru) (pseudokód):

```
aktuální dluh = počáteční dluh;
měs. úroková míra = roční úroková míra / (12 * 100);
For (i = 0; i ≤ počet měsíců hypotéky; i++)
{
        splátka úroku = měs. úroková sazba * aktuální dluh;
        splátka jistiny (úmoru) = měs. splátka úvěru – Splátka úroku;
        If (i == ( počet let držení nemovitosti * 12 / 2 ))
        {
                Střední hodnota splátky úroku =splátka úroku;
                Střední hodnota splátky jistiny = splátka jistiny (úmoru);
        }
        aktuální  dluh −= splátka jistiny (úmoru);
}
```

- Roční růst vlastního jmění [34]:

$$(měs.\,nájem - střední\ hodnota\ splátky\ úroku) * 12\ [Kč]$$

- Výpočet vlastních zdrojů:

$$neobsazenost = měs.\,nájem * 12 * \frac{hodnota\ neobsazenosti}{100}$$

$$první\ rok\ měs.\,nákladů$$
$$= neobsazenst + měs.\,náklady * 12$$
$$+ střední\ hodnota\ splátky\ úroku * 12$$

$$pořizovací\ cena = cena\ nemovitosti - počáteční\ dluh$$

$$vlastní\ zdroje = pořizovací\ cena + první\ rok\ měs.\,nákladů\ [Kč]$$

- Hrubý výnos:

$$\frac{měs.\,nájem * 12}{cena\ nemovitosti} * 100\ [\%]$$

- Roční návratnost (rychlý výpočet):

$$\frac{cena\ nemovitosti}{měs.\,nájem * 12}\ [roky]$$

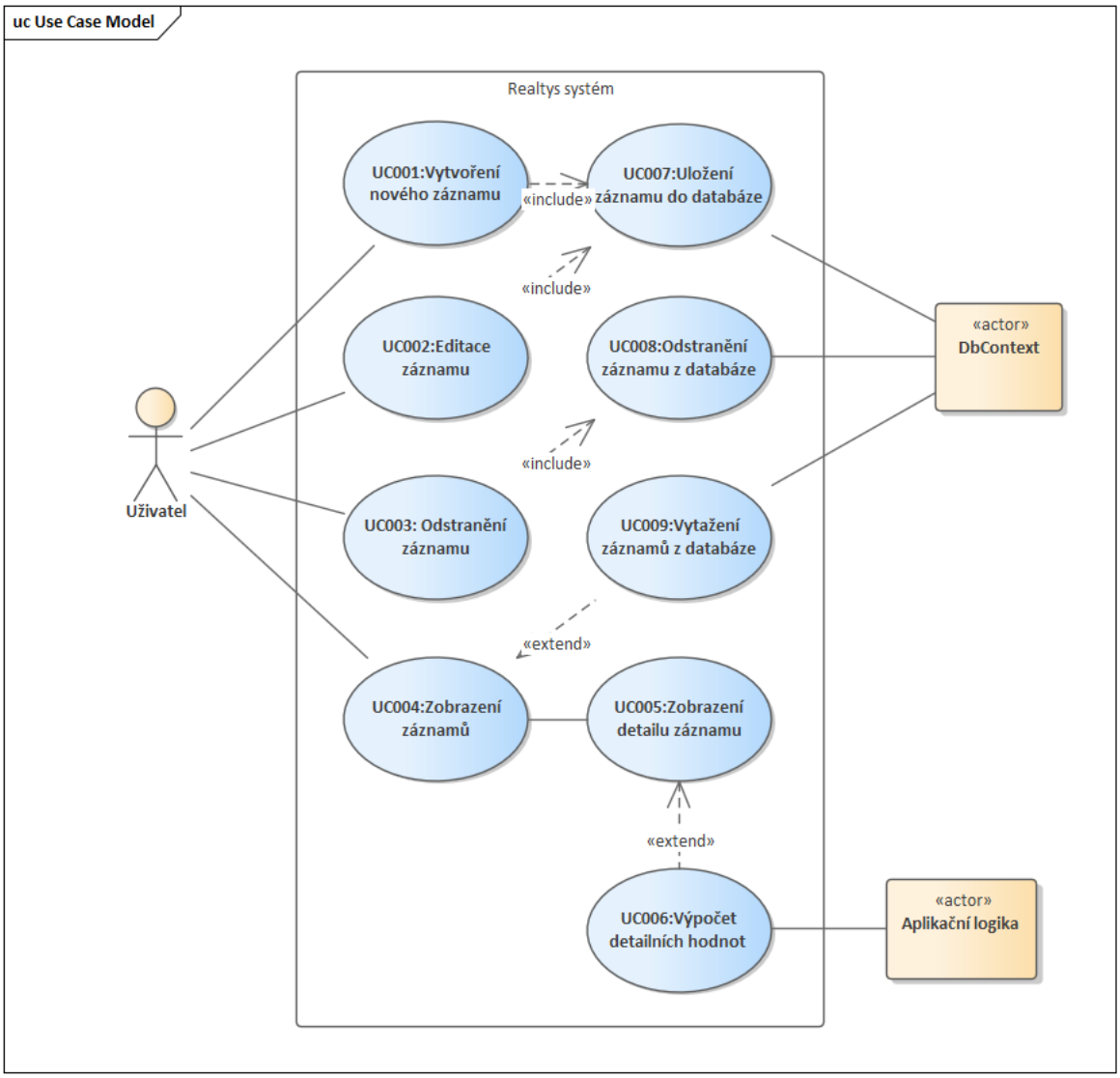
- Roční návratnost vlastních zdrojů [34]:

$$\frac{vlastní\ zdroje}{(měs.\ nájem - střední\ hodnota\ splátky\ úroku) * 12}$$

- Roční zhodnocení vlastních zdrojů [34]:

$$\frac{(měs.\ nájem - střední\ hodnota\ splátky\ úroku) * 12}{vlastní\ zdroje} * 100$$

5 PŘÍPADY UŽITÍ APLIKACE REALTYS



Obrázek 12. Use-case diagram aplikace

Aplikace obsahuje tři aktéry – *Uživatele, DbContext* a *Aplikační logiku*. *Uživatel* přes UI může vytvořit záznam nemovitosti, editovat jej nebo odstranit. S tím mu pomůže *DbContext,* který obsluhuje databázi (v případě aplikace Realtys se jedná o lokální databázi s využitím SQLite) a v ní provádí potřebné změny. *DbContext* také vytahuje záznamy z databáze, které je pak *Uživatel* schopen zobrazit, popřípadě následně editovat nebo odstranit. Pokud *Uživatel* chce zobrazit detail záznamu, *Aplikační logika* pak vypočítá detailní hodnoty zaznamenané nemovitosti.

Scénáře k případům užití jsou v následujících tabulkách:

Název: Vytvoření nového záznamu		
ID: UC001		
Charakteristika: Uživatel chce přidat nový záznam do aplikace.		
Primární aktér: Uživatel		
Vedlejší aktéři: DbContext, Systém		
Vstupní podmínky: DbContext je připojený k databázi.		
Výstupní podmínky: Vytvoření nového záznamu a jeho zápis do databáze, pokud budou splněny všechny validační podmínky.		
Hlavní scénář:		
Krok	Aktér/Systém	Popis
1	Uživatel	Uživatel přejde na záložku pro přidání záznamu.
2	Systém	Systém zobrazí stránku s formulářem.
3	Uživatel	Uživatel vyplní všechny požadované údaje a schválí uložení záznamu.
4	Systém	Systém aplikace provede validaci vstupů.
5	DbContext	DbContext uloží záznam do databáze. [include: UC007]
Výjimky: UC001(4a) – Uživatel chybně vyplní požadovaná pole		

Tabulka 1. Scénář a popis UC001: Vytvoření nového záznamu

Název – Výjimka: Uživatel chybně vyplní požadovaná pole		
ID: UC001(4a)		
Charakteristika: Uživatel chybně vyplnil některé/á pole		
Scénář výjimky:		
Krok	Aktér/Systém	Popis
1	Systém	Systém nalezne validační chybu.
2	Systém	Systém zobrazí chybovou zprávu.

Tabulka 2. Scénář výjimky UC001(4a): Uživatel chybně vyplní požadovaná pole

Název: Editace záznamu		
ID: UC002		
Charakteristika: Uživatel chce editovat již existující záznam.		
Primární aktér: Uživatel		
Vedlejší aktéři: DbContext, Systém		
Vstupní podmínky: Existence záznamu, který bude editován		
Výstupní podmínky: Uložení nových hodnot záznamu do databáze, pokud budou splněny všechny validační podmínky.		
Hlavní scénář:		
Krok	Aktér/Systém	Popis
1	Uživatel	Uživatel přejde na editaci záznamu.
2	Systém	Systém zobrazí stránku s formulářem.
3	Uživatel	Uživatel edituje všechny požadované údaje a schválí uložení záznamu.
4	Systém	Systém aplikace provede validaci vstupů.
5	DbContext	DbContext uloží záznam do databáze. [include: UC007]
Výjimky: UC002(4a) – Uživatel chybně vyplní požadovaná pole		

Tabulka 3. Scénář a popis UC002:Editace záznamu

Název – Výjimka: Uživatel chybně vyplní požadovaná pole		
ID: UC002(4a)		
Charakteristika: Uživatel chybně vyplnil některé/á pole		
Scénář výjimky:		
Krok	Aktér/Systém	Popis
1	Systém	Systém nalezne validační chybu.
2	Systém	Systém zobrazí chybovou zprávu.

Tabulka 4. Scénář výjimky UC002(4a): Uživatel chybně vyplní požadovaná pole

Název: Odstranění záznamu		
ID: UC003		
Charakteristika: Uživatel chce odstranit záznam.		
Primární aktér: Uživatel		
Vedlejší aktéři: DbContext, Systém		
Vstupní podmínky: Existence záznamu, který bude odstraněn.		
Výstupní podmínky: Odstranění záznamu z databáze.		
Hlavní scénář:		
Krok	Aktér/Systém	Popis
1	Uživatel	Uživatel zvolí odstranění záznamu.
2	Systém	Systém se zeptá uživatele, zda opravdu chce záznam odstranit.
3	Uživatel	Uživatel souhlasí s odstraněním.
4	DbContext	DbContext odstraní záznam z databáze. [include: UC008]

Tabulka 5. Scénář a popis UC003:Odstranění záznamu

Název: Zobrazení záznamů		
ID: UC004		
Charakteristika: Zobrazení seznamu záznamů uživateli.		
Primární aktér: Uživatel		
Vedlejší aktéři: DbContext, Systém		
Hlavní scénář:		
Krok	Aktér/Systém	Popis
1	Uživatel	Uživatel zvolí zobrazení seznamu záznamů.
2	DbContext	DbContext předá všechny záznamy systému. [extended by: UC009]
3	Systém	Systém zobrazí všechny uložené záznamy.
Alternativní scénáře: UC004(3a) – V databázi nejsou záznamy		

Tabulka 6. Scénář a popis UC004: Zobrazení záznamů

Název – Alternativní scénář: V databázi nejsou záznamy		
ID: UC004(3a)		
Charakteristika: Databáze neobsahuje záznamy pro zobrazení.		
Alternativní scénář:		
Krok	Aktér/Systém	Popis
1	Systém	Systém zobrazí prázdný seznam.

Tabulka 7. Alternativní scénář UC004(3a): V databázi nejsou záznamy

Název: Zobrazení detailu záznamu		
ID: UC005		
Charakteristika: Zobrazení detailu vybraného záznamu uživateli.		
Primární aktér: Uživatel		
Vedlejší aktéři: Systém, Aplikační Logika		
Vstupní podmínky: Existence záznamu, jehož detail bude zobrazen.		
Výstupní podmínky: Zobrazení detailu záznamu.		
Hlavní scénář:		
Krok	Aktér/Systém	Popis
1	Uživatel	Uživatel zvolí záznam pro zobrazení detailu.
2	Systém	Přejde na zobrazení detailu položky.
3	Aplikační Logika	Aplikační Logika vypočítá potřebné údaje. [extended by: UC006]
4	Systém	Systém zobrazí detail záznamu.

Tabulka 8. Scénář a popis UC005: Zobrazení detailu záznamu

Název: Výpočet detailních hodnot		
ID: UC006		
Charakteristika: Výpočet detailních hodnot.		
Primární aktér: Aplikační Logika		
Vedlejší aktéři: Systém		
Vstupní podmínky: Existence záznamu, jehož hodnoty budou zpracovány.		
Výstupní podmínky: Vypočítání detailních hodnot záznamu.		
Hlavní scénář:		
Krok	Aktér/Systém	Popis
1	Systém	Systém požaduje výpočet hodnot.
2	Aplikační Logika	Aplikační Logika vypočítá potřebné údaje.
3	Systém	Systém předá vypočítané hodnoty.

Tabulka 9. Scénář a popis UC006: Výpočet detailních hodnot

Název: Uložení záznamu do databáze		
ID: UC007		
Charakteristika: Uložení záznamu do databáze.		
Primární aktér: DbContext		
Vedlejší aktéři: Systém		
Vstupní podmínky: Existence databáze, existence a správný formát záznamu		
Výstupní podmínky: Správné uložení do databáze.		
Hlavní scénář:		
Krok	Aktér/Systém	Popis
1	Systém	Systém předá DbContextu záznam pro uložení.
2	Systém	Systém zažádá o uložení záznamu do databáze.
3	DbContext	DbContext uloží záznam do databáze.
4	Systém	Systém přidá uložený záznam mezi ostatní existující záznamy.

Tabulka 10. Scénář a popis UC007: Uložení záznamu do databáze

Název: Odstranění záznamu z databáze		
ID: UC008		
Charakteristika: Odstranění záznamu z databáze.		
Primární aktér: DbContext		
Vedlejší aktéři: Systém		
Vstupní podmínky: Existence databáze, existence a správný formát záznamu		
Výstupní podmínky: Záznam byl odstraněn z databáze.		
Hlavní scénář:		
Krok	Aktér/Systém	Popis
1	Systém	Systém předá DbContextu záznam pro odstranění.
2	Systém	Systém zažádá o odstranění záznamu z databáze.
3	DbContext	DbContext odstraní záznam z databáze.

Tabulka 11. Scénář a popis UC008: Odstranění záznamu z databáze

Název: Vytažení záznamů z databáze		
ID: UC009		
Charakteristika: Vytažení záznamů z databáze		
Primární aktér: DbContext		
Vedlejší aktéři: Systém		
Vstupní podmínky: Existence databáze		
Výstupní podmínky: Vrácení všech záznamů uložených v databázi Systému.		
Hlavní scénář:		
Krok	Aktér/Systém	Popis
1	Systém	Systém požádá DbContext o předání všech záznamů v databázi.
2	DbContext	DbContext vytáhne záznamy z databáze.
3	DbContext	DbContext předá záznamy z databáze Systému.
Alternativní scénáře: UC009(3a) – V databázi nejsou záznamy		

Tabulka 12. Scénář a popis UC009: Vytažení záznamů z databáze

Název – Alternativní scénář: V databázi nejsou záznamy		
ID: UC009(2a)		
Charakteristika: Databáze neobsahuje záznamy pro zobrazení.		
Alternativní scénář:		
Krok	Aktér/Sys-tém	Popis
1	DbContext	DbContext nenajde v databázi žádný záznam
2	DbContext	DbContext předá prázdnou množinu Systému.

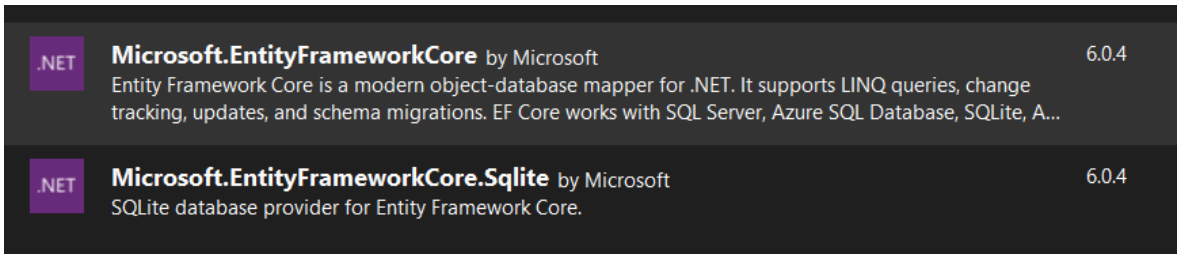
Tabulka 13. Alternativní scénář UC009(2a): V databázi nejsou záznamy

6 ZABEZPEČENÍ APLIKACE

Aplikace je podle hrozících útoků zabezpečena. Jelikož aplikace nekomunikuje se externím serverem a nekomunikuje přes Internet, nehrozí tedy útok Man-in-the-middle a komunikace se tedy nemusí nijak šifrovat. Databáze byla zvolena lokální v podobě SQLite. Jelikož je aplikace spíše forma poznámkových záznamu, není potřeba ani autentizace uživatele.

6.1 Ošetření SQL Injection

SQL Injection je v aplikaci ošetřeno pomocí Entity Framework Core (viz. Obrázek 13), což je ORM knihovna od Microsoftu.



Obrázek 13. EntityFrameworkCore a EntityFrameworkCore.Sqlite packages

V aplikaci jsou vytvořeny dva modely, které jsou pak namapovány na tabulky v databázi. S těmito modely je následně pracováno v celé aplikaci. Příklad kódu objektového modelu *RealEstate* z aplikace, představující model nemovitosti:

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Realtys.Models;

    /// <summary>  Model nemovitosti.
    [Table(nameof(RealEstate))]
    public class RealEstate
    {
        /// <summary> ID nemovitosti. Primární klíč do databáze.
        [Key]
        [Required]
        public int ID { get; set; }
        /// <summary> Název nemovitosti, jak bude uložena v databázi.
        [StringLength(255)]
        [Required]
        public string Name { get; set; }
        /// <summary> Měsíční náklady na nemovitost.
        [Required]
        public int? MonthlyExpenses { get; set; }
        /// <summary> Mesíční nájem z nemovitosti.
        [Required]
        public int? MonthlyRent { get; set; }
        /// <summary> Cena nemovitosti.
        [Required]
        public int? RealtyPrice { get; set; }
```

```
        /// <summary> Neobsazenost nemovitosti za první rok v %.
        [Required]
        public double? Vacancy { get; set; }
        /// <summary> Na jak dlouho bude nemovitost držena.
        [Required]
        public int? ForYears { get; set; }
        /// <summary> Použití úvěru.
        [Required]
        public bool MortgageUsage { get; set; }
    }
```

Za pomoci *DataAnnotations* je definováno mapování do databáze jednotlivých properties:

- *[Table(nameof(RealEstate))]* – představuje název tabulky v databázi, konkrétně to může být název třídy (v tomto případě *RealEstate*)
- *[Key]* – je anotace pro primární klíč (pro foreign key je používaná anotace s názvem třídy modelu, na kterou se klíč odkazuje *[ForeignKey(nameof(RealEstate))])*
- *[Required]* – namapuje do databáze, že hodnota nesmí být null (v databázi jako NOT NULL)
- *[StringLength(255)]* – tato anotace určuje maximální velikost znakového řetězce

Dále právě pomocí Entity Frameworku je v aplikaci definován DbContext, který v celé aplikaci pak slouží jako můstek mezi databází a daty aplikace.

```
using Microsoft.EntityFrameworkCore;
using Realtys.Models;

namespace Realtys.Database;
public class RealtysDbContext : DbContext
{
    public DbSet<RealEstate> RealEstates { get; set; }
    public DbSet<Mortgage> Mortgages { get; set; }


    public RealtysDbContext(DbContextOptions options) : base(options)
    {
    }

}
```

Instance DbContextu je pak definována v souboru MauiProgram.cs kde je definován i typ databáze (v tomto případě lokální databáze SQLite (viz. Obrázek 13)):

```
builder.Services.AddDbContext<RealtysDbContext>(options =>
                options.UseSqlite("Data source=" + PATH)
                );
```

Kde *PATH* je kombinace speciální složky *LocalApplicationData* a názvu souboru databáze (v tomto případě *Realtys.db*):

```
      private static string PATH = Path.Combine
            (
                  Environment.GetFolderPath
                        (
```

```
                                Environment.SpecialFolder.LocalApplicationData
                            ),
                        "Realtys.db"
                    );
```

Aby se mohl DbContext použít, je potřeba ho pomocí Dependency Injection vložit přes konstruktor třídy a pak je jej možné používat.

```
public partial class App : Application
{
    public static RealtysDbContext DbContext;


    public App(RealtysDbContext dbContext)
    {
        DbContext = dbContext;
        InitializeComponent();
        MainPage = new AppShell();
    }

}
```

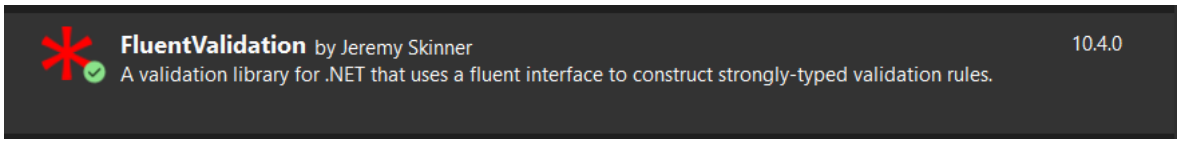
Pak jej jednoduše použít pro komunikaci s databází a jejími entitami. Příklady dotazů, za pomoci Entity Frameworku, použitých v aplikaci:

```
var realties = DbContext.RealEstates.ToList();//vytažení všech záznamů tabulky
var realEstate = App.DbContext.RealEstates.FirstOrDefault(r => r.ID == id);
//vytažení jednoho záznamu tabulky podle ID
DbContext.RealEstates.Add(this.RealEstate); //přidání záznamu do databáze
await DbContext.SaveChangesAsync(); //uložení změn do databáze
```

V aplikaci dále není použito ručně psaného dotazu, tudíž je tímto zabezpečena proti SQL Injection.

6.2 Validace vstupů

Pro validaci vstupních dat bylo zvoleno FluentValidation (viz. Obrázek 14) kde si programátor přesně stanoví, co chce pro danou položku validovat.



Obrázek 14. FluentValidation package

Třída pro validaci pak dědí od třídy AbstractValidator kde se do <*T*> předá typ, který se bude validovat.

```
using FluentValidation;
using Realtys.Models;
```

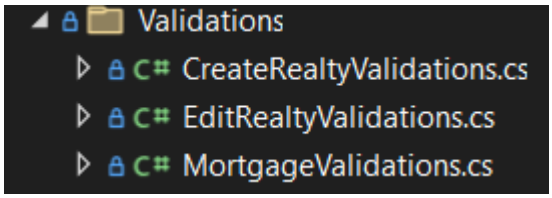
```
public class CreateRealtyValidations : AbstractValidator<RealEstate>
```

Konkrétní validační pravidla pak vypadají následovně:

```
//Pravidlo pro Měsíční nájem
RuleFor(x => x.MonthlyRent)
        .NotEmpty().WithMessage("Měsíční nájem je povinný údaj.")
        .GreaterThanOrEqualTo(0).WithMessage("Měsíční nájem musí být >= 0.");

//Pravidlo pro Název nemovitosti
RuleFor(x => x.Name).NotEmpty().WithMessage("Název nemovitosti je požadován.");
```

Pro většinu validací je v aplikaci použita metoda *NotEmpty* a metoda testující interval pro zadané číslo. Například validace pro Měsíční nájem (*MonthlyRent*) pravidla otestují, zda se jedná o null, prázdný string nebo se skládá pouze z white-space znaků a pokud ano, tak vyhodí error s předdefinovanou zprávou. Pokud vstup projde přes první metodu, otestuje se, zda se jedná o číslo v požadovaném intervalu. Pravidlo pro Název nemovitosti (*Name*) pak testuje jestli není vstup null, prázdný nebo složeny z white-space znaků. Pokud není podmínka splněna, vyhodí error s definovanou zprávou.[33]

Aplikace pak používá tři validační třídy (viz. Obrázek 15). První a druhá je pro vytvoření a editaci záznamu nemovitosti a třetí validační třída je pro úvěr (vytvoření i editace).



Obrázek 15. Validační třídy aplikace

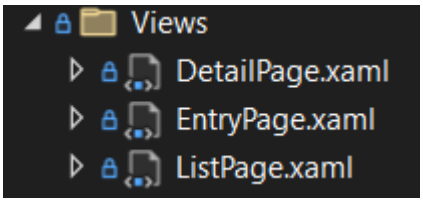
6.3 Použité API

Při tvorbě aplikace byly vybírány pouze ty API, které jsou buď přímo od Microsoftu (.NET, EntityFrameworkCore) a ty, které jsou hojně používány programátory a stále aktualizovány jejich tvůrci (FluentValidation [33]) nebo od firmy Syncfusion s placenými API, která poskytuje zdarma licenci do obratu $1mil. a do pěti vývojářů v týmu (Syncfusion.Maui Charts [39] a Sliders [38]). Jediná možná riskantní API může být .NET MAUI, která je v době psaní práce ve fázi Preview a může obsahovat nějaké chyby.

7 REALIZECE A POPIS KLÍČOVÝCH ČÁSTÍ APLIKACE

7.1 Views aplikace

Aplikace má tři základní Views(viz. Obrázek 16). Každá prezentuje jinou část aplikace. Dále pak má View, které definuje Shell rozložení aplikace, s názvem AppShell.xaml.



Obrázek 16. Základní Views aplikace

7.1.1 AppShell

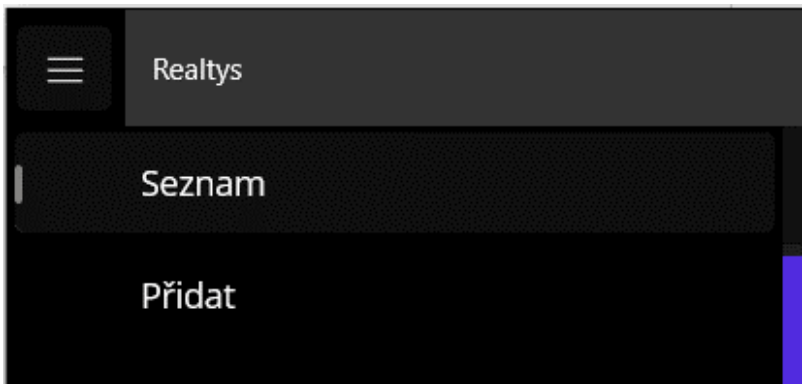
AppShell je View, které oproti ostatním Views nemá base třídu ContentPage, ale Shell. To nám umožní použití tzv. Hamburger Menu v aplikaci (viz. Obrázek 17).

```
public partial class AppShell : Shell

                    vs.

public partial class ListPage : ContentPage
```

V AppShell.xaml:

```
<Shell xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
       xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
       xmlns:local="clr-namespace:Realtys.Views"
       x:Class="Realtys.AppShell"
       Title="Realtys"
       FlyoutBackgroundColor="Black"
       BackgroundColor="{DynamicResource PrimaryColor}"
       ForegroundColor="{DynamicResource SecondaryColor}">

    <FlyoutItem Title="Seznam"
                Route="first">

        <ShellContent ContentTemplate="{DataTemplate local:ListPage}"/>

    </FlyoutItem>

    <FlyoutItem Title="Přidat"
                Route="Entry">

        <ShellContent ContentTemplate="{DataTemplate local:EntryPage}"/>

    </FlyoutItem>

</Shell>
```

V *FlyoutItem* je nastavena Route, na kterou se pak odkazuje v aplikaci. V ShellContent je pak View, které chceme zobrazit po kliknutí na položku v Hamburger Menu (viz. Obrázek 17).



Obrázek 17. Hamburger Menu

7.1.2 View Seznam nemovitostí

První View je Seznam nemovitostí (*ListPage.xaml*). V tomto View jsou prezentovány uložené záznamy pomocí CollectionView. *x:Name* určuje název elementu, *SelectionMode* je nastaven na hodnotu *Single*, protože je cílem zvolit pouze jeden záznam. Dále je v *CollectionView.ItemsLayout* nastaveno zobrazení jednoho prvku na řádek. Obsah pole je pak definován v *CollectionView.ItemTemplate*.

```
<CollectionView x:Name="listView"
                SelectionMode="Single"
                BackgroundColor="{DynamicResource BackgroundColor}">

        <CollectionView.ItemsLayout>
            <GridItemsLayout Span="1" Orientation="Vertical"/>
        </CollectionView.ItemsLayout>
        <CollectionView.ItemTemplate>
                    .
                    .
                    .

        </CollectionView.ItemTemplate>
</CollectionView>
```

Naplnění CollectionView probíhá v souboru ListPage.xaml.cs v metodě OnAppearing následovně:

```
protected override void OnAppearing()
{
    base.OnAppearing();
    var realties = DbContext.RealEstates.ToList();
    realties.Reverse();
    listView.ItemsSource = realties;
    listView.SelectedItem = null;
```

```
}
```

Aby proklik správně fungoval, musí se do EventHandler-u *CollectionView.SelectionChanged* přiřadit metoda, která se zavolá při kliknutí na položku.

Přiřazení metody *OnCollectionViewSelectionChanged* do EventHandler-u *SelectionChanged* v konstruktoru *ListPage:*

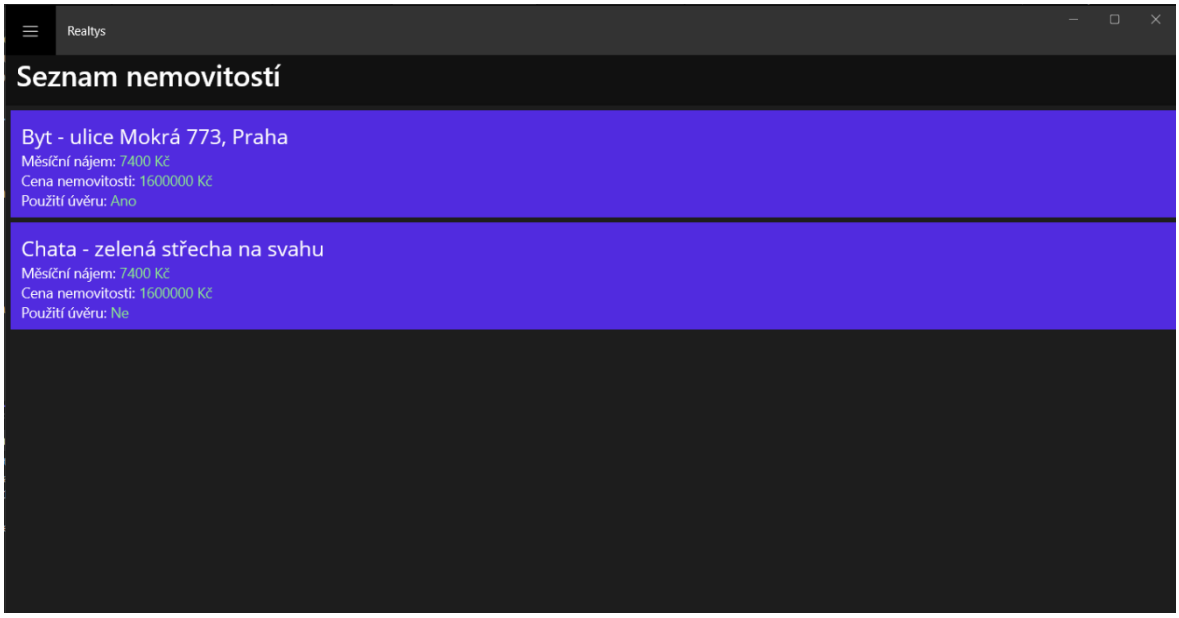
```
public ListPage(RealtysDbContext dbContext)
{
    InitializeComponent();
    DbContext = dbContext;
    listView.SelectionChanged += OnCollectionViewSelectionChanged;
}
```

Přiřazená metoda *OnCollectionViewSelectionChanged* kde:

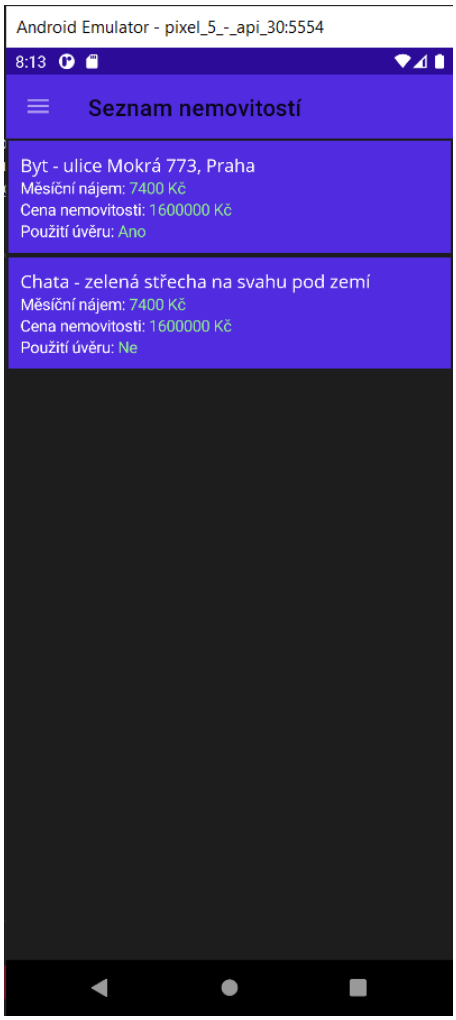
```
async void OnCollectionViewSelectionChanged
      (object sender, SelectionChangedEventArgs e)
{
    var item = e.CurrentSelection.FirstOrDefault();
    if (item != null)
    {
       await Navigation.PushAsync(new DetailPage()
       {
            BindingContext = new DetailViewModel((item as RealEstate).ID)
       });
    }
}
```

V metodě *OnCollectionViewSelectionChanged* lze vidět přiřazení *DetailViewModel-u* do *BindingContext-u* View *DetailPage.*

Takto vypadá stránka *Seznam nemovitostí* při zpuštění aplikace (Windows - Obrázek 18, Android - Obrázek 19):



Obrázek 18. Uživatelský pohled na *ListPage* – Windows



Obrázek 19. Uživatelský pohled na *ListPage* – Android

Aby se hodnota u *Použití úvěru:* zobrazila jako Ano/Ne je ve View použit *Converter* při propojení hodnoty, který převádí boolean hodnotu true/false na string „Ano“/„Ne“:

```
Text="{Binding MortgageUsage, Converter={StaticResource boolToString}}"
```

Converter je potřeba definovat na začátku View:

```
    <ContentPage.Resources>
        <converter:BoolToStringConverter x:Key="boolToString" />
    </ContentPage.Resources>
```

7.1.3 View pro vytvoření a editaci záznamu

Pro přidání a editaci záznamu slouží stejné View s formulářem *EntryPage.xaml*. Do tohoto View se pomocí *BindingContext-u* předá záznam do ViewModelu pro editaci, jinak jsou ve ViewModelu připraveny properties pro vytvoření nového záznamu. Kód pro View obsahuje několik Entry elementů, do kterých jsou uživatelem psány vstupy:

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

    <ScrollView BackgroundColor="{DynamicResource BackgroundColor}">
        <StackLayout Margin="10">
            <Label  x:Name="errors"
                    TextColor="Red"
                    Text="{Binding EditCreateErrors}"/>

            <Label Text="Nemovitost: " FontSize="Title"/>
            <Label Text="Pojmenování nemovitosti: "/>
            <Entry Placeholder="Vložte název položky"
                   Text="{Binding RealEstate.Name}"
                   HeightRequest="50"
                   x:Name="nameEntry"/>

            <Label Text="Cena nemovitosti: "/>
            <Entry Keyboard="Numeric"
                   Placeholder="Cena nemovitosti"
                   Text="{Binding RealEstate.RealtyPrice,
                         Converter={StaticResource intToString}}"
                   HeightRequest="50"
                   x:Name="priceEntry"/>
                   .
                   .
                   .

            <Grid ColumnDefinitions="110,auto,*">
                <Label Grid.Column="0" Padding="0,5,0,0">Přidat úvěr: </Label>
                <CheckBox x:Name="addMortgageCheckBox"
                          CheckedChanged="OnCheckBoxCheckedChanged"
                          Grid.Column="1"/>
            </Grid>

            <Grid x:Name="mortgageGrid"
                  IsVisible="{Binding IsMortgageUsed}"
                  RowDefinitions="Auto,Auto,Auto,Auto,Auto,Auto,Auto">
                <Grid.ColumnDefinitions>
```

```
                    <ColumnDefinition Width="*"/>
                </Grid.ColumnDefinitions>

                <Label Text="Úvěr: " FontSize="Header" Grid.Row="0"/>

                <Label Text="Roční úroková míra (v %)" Grid.Row="1"/>
                <Entry x:Name="interestEntry"
                       Keyboard="Numeric"
                       Grid.Row="2"
                       Placeholder="Roční úroková míra v %"
                       Text="{Binding Mortgage.Interest,
                              Converter={StaticResource doubleToString}}"
                       HeightRequest="50"/>
                                .
                                .
                                .
            </Grid>

            <Grid ColumnDefinitions="*,*">
                <Button Text="Save"
                        Grid.Row="1"
                        Grid.Column="1"
                        VerticalOptions="Start"
                        Command="{Binding SaveCommand}"/>
            </Grid>
        </StackLayout>
    </ScrollView>
</ContentPage>
```

Label na začátku je spojen se zprávami při neprojití vstupů validací.

První *Entries* jsou pro zadání hodnot k nemovitosti. První *Entry* je pro vstup Názvu nemovitosti, druhé je pak příklad *Entry* pro zadání číselné hodnoty, a proto byla klávesnice nastavena na *Numeric*, aby se na mobilním zařízení použila číselná klávesnice. K číselným *Entries* je pak přiřazen konvertor, který vstupní text převede na číslo nebo null hodnotu (pokud bylo uživatelem zadány znaky nereprezentující čísla). Dále následuje *Grid* obsahující *CheckBox* pro přidání úvěru k nemovitosti. Pokud je *CheckBox* zaškrtnut, zobrazí se *DisplayAlert* pro uživatele, zda chce přidat úvěr k nemovitosti (viz. Obrázek 22), popř. jej odebrat pokud již byl *CheckBox* zaškrtnut nebo záznam obsahoval úvěr (případ editace záznamu), kde se v *DisplayAlert-u* systém dotáže, zda opravdu chce uživatel odebrat záznam úvěru s tím, že již vyplněné hodnoty ztratí (viz. Obrázek 23). Zobrazení *DisplayAlert-u* probíhá v metodě na kterou je *CheckBox* připojen *EventHandler-em CheckedChanged*: *OnCheckBoxCheckedChanged* v souboru *EntryPage.xaml.cs*.

```
async void OnCheckBoxCheckedChanged(object sender, CheckedChangedEventArgs e)
        {
            addMortgageCheckBox.CheckedChanged -= OnCheckBoxCheckedChanged;
            var viewModel = (EditCreateViewModel)BindingContext;

            if (!mortgageGrid.IsVisible)
            {
                bool check = await DisplayAlert(
```

```
                "Přidání úvěru",
                "Bude přidán úvěr k nemovitosti",
                "OK",
                "Cancel"
                );

            if (check)
            {
                viewModel.IsMortgageUsed = check;
            }
            addMortgageCheckBox.IsChecked = check;
        }
        else
        {
            bool check = await DisplayAlert(
                "Odebrání úvěru",
                "Bude odebrán úvěr k nemovitosti,
                 veškerý vyplněný obsah, bude ztracen!",
                "OK",
                "Cancel"
                );

            if (check)
            {
                viewModel.IsMortgageUsed = !check;
                viewModel.Mortgage = new Mortgage();
            }
            addMortgageCheckBox.IsChecked = !check;

        }
        addMortgageCheckBox.CheckedChanged += OnCheckBoxCheckedChanged;
    }
```

Metoda je na začátku z *EventHandler-u* oddělána kvůli případnému odškrtnutí/zaškrtnutí, které by znovu vyvolalo událost, která metodu zavolá. Po proběhnutí celé metody se *EventHandler-u* znovu přiřadí.

Jako další je *Grid*, který zobrazuje formulář pro úvěr k nemovitosti. Tento formulář se zobrazí/skryje ihned po tom, co uživatel klikne na *CheckBox*. Pod celým formulářem je pak tlačítko na uložení záznamu, které je přes *Command* atribut napojeno na *Command* ve ViewModelu, jenž ukládá změny nebo celý nový záznam do databáze.

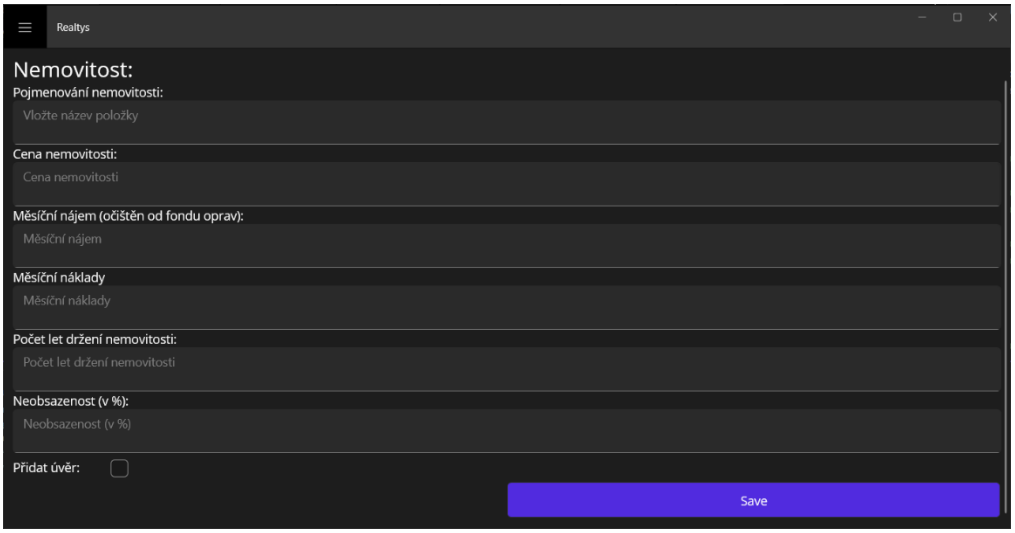
Zobrazení View uživateli:

Windows

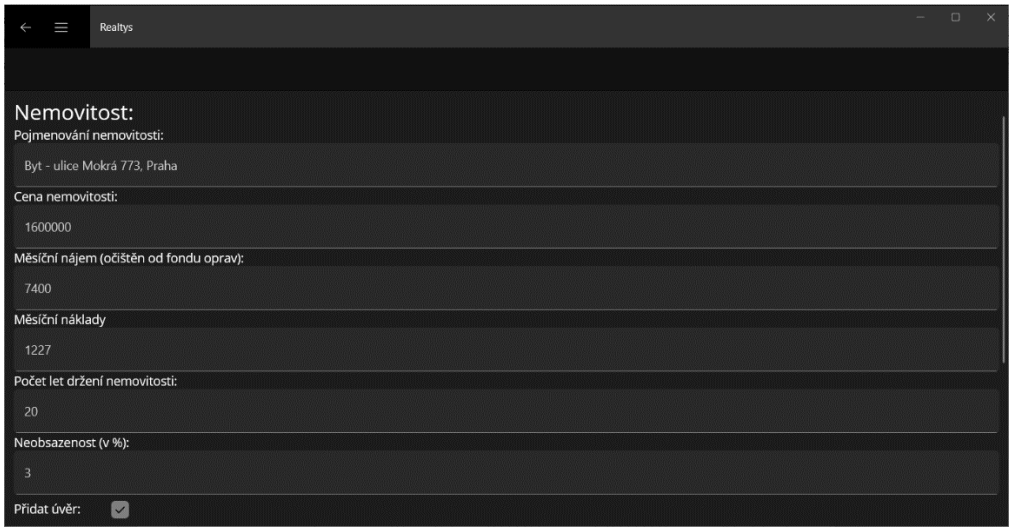
- Vytvoření viz. Obrázek 20, Obrázek 24 a Obrázek 25
- Editace viz. Obrázek 21
- *DisplayAlert* viz. Obrázek 22 a Obrázek 23

Android

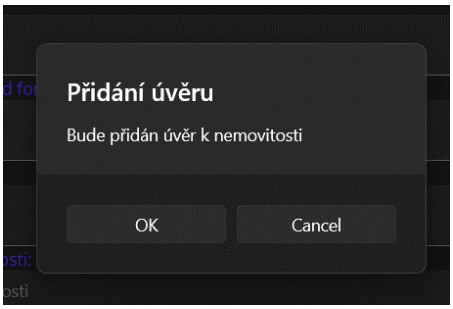
- Vytvoření viz. Obrázek 26



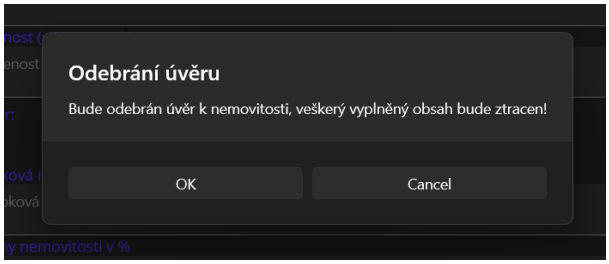
Obrázek 20. Formulář pro vytvoření nemovitosti – Windows



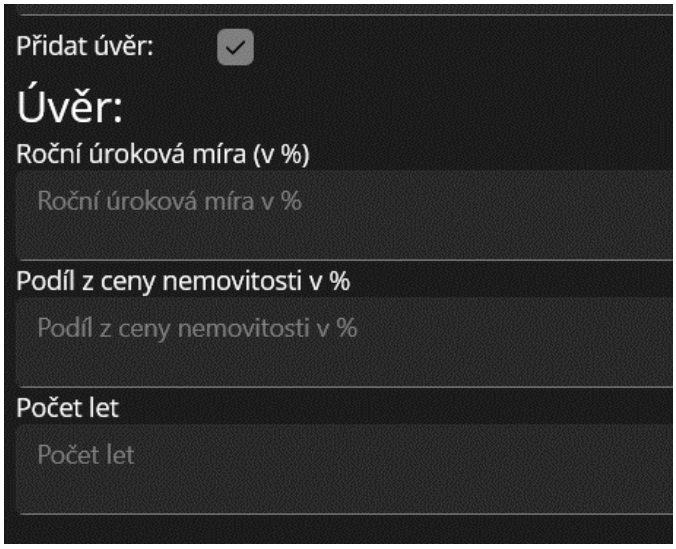
Obrázek 21. Formulář s předvyplněnými daty pro editaci



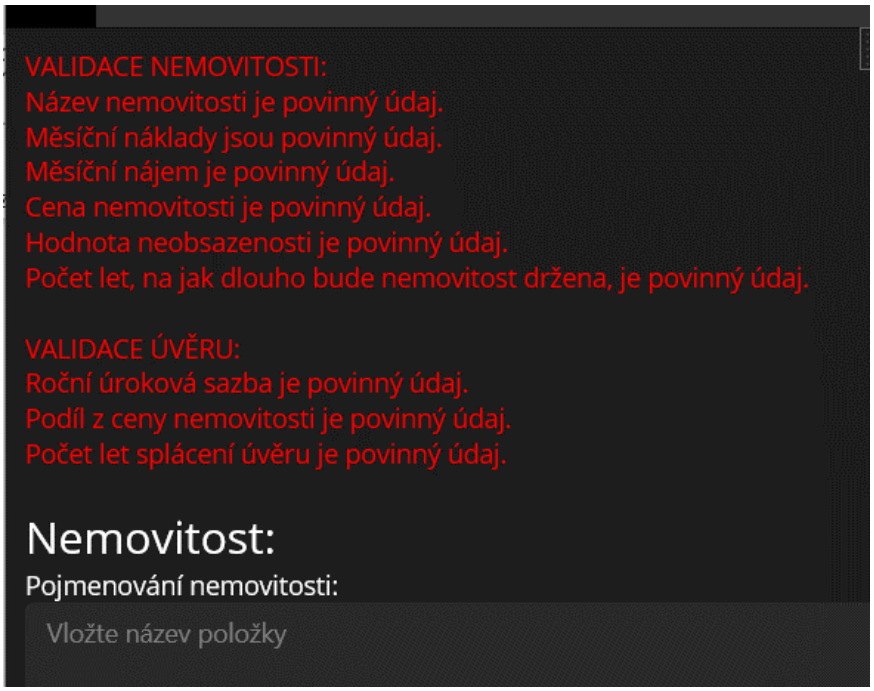
Obrázek 22. *DisplayAlert* pro přidání úvěru



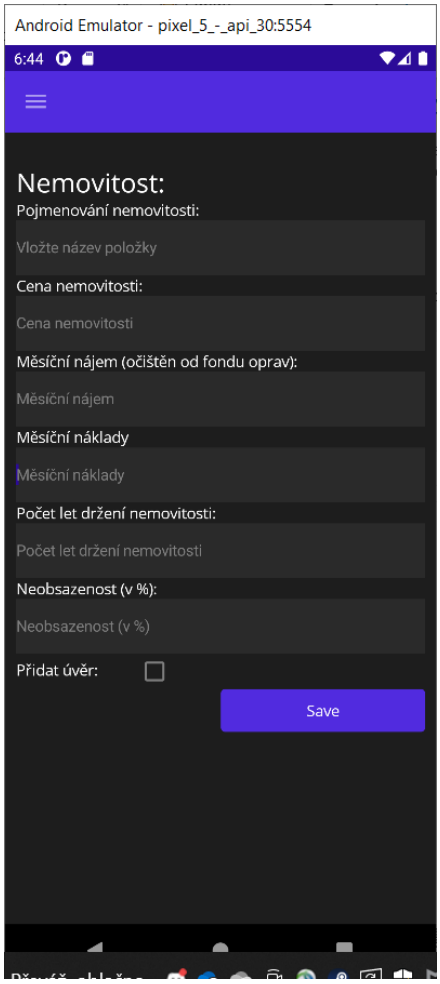
Obrázek 23. *DisplayAlert* pro odebrání úvěru



Obrázek 24. Formulář pro úvěr



Obrázek 25. Formulář s validačními errory



Obrázek 26. Formulář pro vytvoření nemovitosti – Android

7.1.4 View Detail nemovitosti

Toto View obsahuje detail nemovitosti a je definováno souborem *DetailPage.xaml* a využívá ViewModel *DetailViewModel.cs*. Zobrazuje uživateli detailní výpočty záznamu, jako je například návratnost vlastních zdrojů nebo roční růst vlastního jmění. Tyto údaje jsou klíčové v rozhodnutí investora (uživatele), zda se jedná o výhodnou, či nevýhodnou investici. Záznam je zvolen ze seznamu nemovitostí a detailní výpočty pak pro View vypočítá ViewModel z uložených hodnot konkrétního předaného záznamu.

```
<Label Margin="0,0,0,0"
       Text="{Binding RealEstate.Name}"
       FontSize="Large"
       HorizontalTextAlignment="Start"/>
<Label x:Name="realtyDetail"
       VerticalOptions="Center"
       HorizontalOptions="StartAndExpand"
       TextColor="{DynamicResource SecondaryColor}">
    <Label.FormattedText>
        <FormattedString>
            <FormattedString.Spans>
```

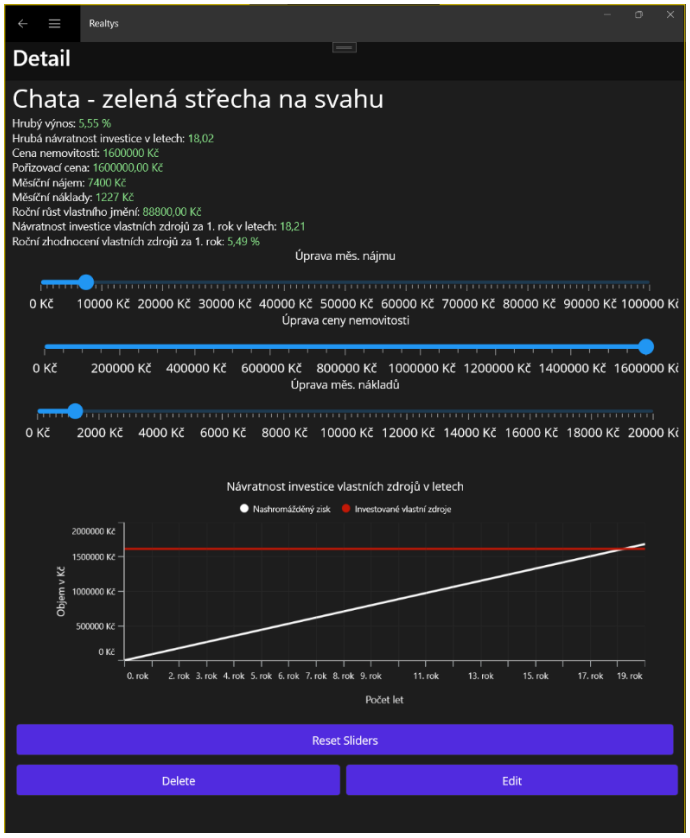
```
                <Span Style="{StaticResource whiteSpan}"
                      Text="Hrubý výnos: "/>
                <Span Style="{StaticResource greenSpan}"
                      Text="{Binding HrubyVynos, StringFormat='{0:F2}'}" />
                <Span Style="{StaticResource greenSpan}" Text=" % &#10;"/>
                                .
                                .
                                .
            </FormattedString.Spans>
        </FormattedString>
    </Label.FormattedText>
</Label>

<Label Margin="10,0,0,0"
       IsVisible="{Binding RealEstate.MortgageUsage}"
       Text="Úvěr k nemovitosti:"
       FontSize="Subtitle"/>
<Label Margin="10"
       x:Name="mortgageDetail"
       VerticalOptions="Center"
       HorizontalOptions="StartAndExpand"
       IsVisible="{Binding RealEstate.MortgageUsage}"
       TextColor="{DynamicResource SecondaryColor}">
    <Label.FormattedText>
        <FormattedString>
            <FormattedString.Spans>
                <Span Style="{StaticResource whiteSpan}"
                      Text="Počáteční dluh: " />
                <Span Style="{StaticResource greenSpan}"
                      Text="{Binding PocatecniDluh, StringFormat='{0:F2}'}" />
                                .
                                .
                                .
            </FormattedString.Spans>
        </FormattedString>
    </Label.FormattedText>
</Label>
<Grid ColumnDefinitions="*,*" RowDefinitions="auto,auto">
    <Button Text="Reset Sliders"
            Grid.Row="0" Grid.ColumnSpan="2"
            Command="{Binding ResetCommand}"
            Margin="5"/>
    <Button Text="Delete"
            Grid.Row="1" Grid.Column="0"
            VerticalOptions="Center"
            Command="{Binding DeleteCommand}"
            Margin="5"/>
    <Button Text="Edit"
            Grid.Row="1" Grid.Column="1"
            VerticalOptions="Center"
            Command="{Binding EditCommand}"/>
      </Grid>
```

V tomto View jsou převážně jen výpisy hodnot. Je zde použití *FormatedString-u* a v něm jsou dané hodnoty nabindované na properties ve ViewModelu. Hodnota některých *Span-ů* je *
* , což značí znak konce řádku. Do *StringFormat* se pak zadává formát výpisu reálných čísel. Pokud záznam obsahuje úvěr, tak je zobrazen i *Label* pro tento úvěr pomocí atributu *IsVisible* napojeného na hodnotu ve ViewModelu. Dále View obsahuje *Slidery*,

které slouží pro interakci s hodnotami a tím si upravit výstupní hodnoty [38] a graf, který graficky zobrazuje změnu hodnot [39].

Dále jsou ve View tři tlačítka. První je pro resetování sliderů, druhé je pro odstranění záznamu a třetí pro editaci. Každé tlačítko je pak spojeno s příslušným *Command-em* ve ViewModelu. Níže je pak demonstrace, jak View vypadá pro uživatele (Windows viz. Obrázek 27 a Obrázek 28, Android viz. Obrázek 29 a Obrázek 30).

Obrázek 27. Zobrazení detailu nemovitosti s úvěrem - Windows



Obrázek 28. Zobrazení detailu nemovitosti – Windows

Obrázek 29. Zobrazení detailu nemovitosti s úvěrem - Android

Obrázek 30. Zobrazení detailu nemovitosti - Android

7.2 Modely aplikace

Aplikace obsahuje dva modely. Model nemovitosti (*RealEstate*), který představuje nemovitost a obsahuje properties vztahující se právě k nemovitosti (např. Cena nemovitosti nebo Nájem z nemovitosti). Model hypotečního úvěru (*Mortgage*) pak obsahuje properties vztahující se k úvěru jako je Úroková sazba nebo Velikost splátky úvěru.

Část kódu modelu *RealEstate*:

```
/// <summary> Model nemovitosti.
[Table(nameof(RealEstate))]
public class RealEstate
{
    [NotMapped]
    private int? monthlyExpenses;
    [NotMapped]
    private int? monthlyRent;
    [NotMapped]
    private int? realtyPrice;
    [NotMapped]
    private double? vacancy;
    [NotMapped]
    private int? forYears;
    /// <summary> ID nemovitosti. Primární klíč do databáze.
    [Key]
    [Required]
    public int ID { get; set; }
    /// <summary> Název nemovitosti, jak bude uložena v databázi.
```

```
        [StringLength(255)]
        [Required]
        public string Name { get; set; }
        /// <summary> Měsíční náklady na nemovitost.
        [Required]
        public string MonthlyExpenses
        {
            get => monthlyExpenses.ToString();
            set
            {
                if (Int32.TryParse(value, out int i)) monthlyExpenses = i;
                else monthlyExpenses = null;
            }
        }
                    .
                    .
                    .
        /// <summary> Použití úvěru.
        [Required]
        public bool MortgageUsage { get; set; }
    }
```

Model obsahuje property:

- ID záznamu (*ID*); typ integer
- Název nemovitosti (*Name*); typ string
- Měsíční náklady (*MonthlyExpenses*); typ string, spojený s nullable integer fieldem *monthlyExpenses*
- Měsíční nájem (*MonthlyRent*); typ string, spojený s nullable integer fieldem *monthlyRent*
- Cena nemovitosti (*RealtyPrice*); typ string, spojený s nullable integer fieldem *realtyPrice*
- Neobsazenost nemovitosti (*Vacancy*); typ string, spojený s nullable double fieldem *vacancy*
- Počet držení nemovitosti (*ForYears*); typ string, spojený s nullable integer fieldem *forYears*
- Použití úvěru (*MortgageUsage*); typ boolean

Část kódu modelu *Mortgage:*

```
/// <summary> Model hypotečního úvěru.
[Table(nameof(Mortgage))]
public class Mortgage
{
    [NotMapped]
    private int? forYears;
    [NotMapped]
    private double? share;
```

```
        [NotMapped]
        private double? interest;
        /// <summary> ID úvěru.
        [Key]
        public int ID { get; set; }
        /// <summary> Roční úroková sazba úvěru.
        [Required]
        public string Interest
        {
            get => interest.ToString();
            set
            {
                if(Double.TryParse(value, out double i)) interest = i;
                else interest = null;
            }
        }
                    .
                    .
                    .
        /// <summary> Splátka úvěru.
        [Required]
        public double Payment { get; set; }
        /// <summary> ID nemovitosti, ke které je úvěr vázán.
        [ForeignKey(nameof(RealEstate))]
        public int RealtyID { get; set; }

}
```

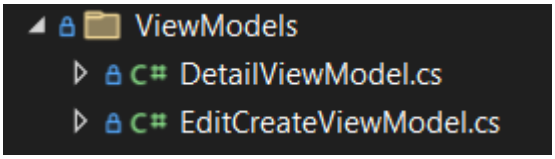
Model obsahuje property:

- ID záznamu (*ID*); typ integer
- Roční úroková sazba (*Interest*); typ string, spojený s nullable double fieldem *interest*
- Podíl z ceny nemovitosti (*Share*); typ string, spojený s nullable double fieldem *share*
- Počáteční dluh (*InitialDebt*); typ double
- Počet držení nemovitosti (*ForYears*); typ string, spojený s nullable integer fieldem *forYears*
- Splátka úvěru (*Payment*); typ double
- ID nemovitosti (*RealtyID*); typ integer

Modely sice používají číselné hodnoty a jejich použití je ztíženo typem string, ale pro vstupy při vytváření a editaci bylo potřeba použít tento typ kvůli správné funkci.

7.3 ViewModely aplikace

Aplikace obsahuje dva ViewModely (viz. Obrázek 31). *DetailViewModel.cs* slouží k detailním výpočtům záznamu a *EditCreateViewModel.cs* obsahuje zpracování dat pro vytvoření a editaci záznamů.



Obrázek 31. ViewModely aplikace

DetailViewModel obsahuje metody, které demonstrují vzorce z blogu AdolMonitor [34]. Výsledky po dosazení do těchto vzorců předá do View *DetailPage,* kde jsou poté prezentovány uživateli.

EditCreateViewModel pracuje s ukládáním změn nebo nového záznamu do databáze, dále obsahuje validaci vstupů a kontroluje, zda byl přidán úvěr k záznamu nemovitosti (popř. zda byl odebrán při editaci a jeho následné odstranění z databáze). V tomto ViewModelu je používána událost *PropertyChanged*, která hlídá změnu property, na kterou je aplikována. V případě aplikace je *PropertyChanged* aplikována na properties, které budou přidávány nebo editovány, a které zároveň reprezentují záznamy v databázi. Aby mohla třída využít tuto událost, musí použít rozhraní *INotifyPropertyChanged* [36]

```
public class EditCreateViewModel : INotifyPropertyChanged
```

Demonstrace použití *PropertyChanged EventHalndler-u*:

```
public Mortgage Mortgage
{
    get { return _Mortgage; }
    set
    {
        _Mortgage = value;
        OnPropertyChanged(nameof(Mortgage));
    }
}

protected void OnPropertyChanged(string name)
{
    PropertyChangedEventHandler handler = PropertyChanged;

    if (handler != null)
    {
        handler(this, new PropertyChangedEventArgs(name));
    }
}
```

8 ZÁVEREČNÉ ZHODNOCENÍ .NET MAUI

Technologie .NET MAUI je při realizaci této práce stále ve fázi Preview, ale i přes tuto limitaci je to vhodná volba pro vývojáře, kteří mají zkušenosti s vývojem v Xamarin.Forms nebo WPF. V současné době již existuje i Release Candidate, tudíž by měl být framework stabilní. Framework .NET MAUI přináší ovšem možnost tvorby aplikace v Blazoru tím, že hostuje Blazor View v aplikaci. To pomáhá i programátorovi v Blazoru, který dokáže udělat multiplatformního nativního klienta své webové aplikace na mobilních i desktopových platformách. Prakticky se tedy hotový kód aplikace vytvořen v Xamarin.Forms nebo Blazor pouze přesune do .NET MAUI projektu a aplikace by měla fungovat. Další plus spočívá v použití pouze jedné kódové základny pro všechny platformy, kdy se naprostá většina kódu sdílí napříč platformami. Pro nastavení sdílených částí, jako například ikon, pak programátorovi stačí upravit soubor projektu (.csproj), kde jsou zvýrazněny části, které vývojář většinou upravuje (viz. kód níže). Co se týče ikon aplikace, .NET MAUI, oproti například Ionic Frameworku, nepotřebuje všechny velikosti ikony nahrát do souborů aplikace, ale stačí jedna velikost .svg ikony a posléze definovat cestu k ikoně v .csproj souboru.

```
<!-- App Icon -->
<MauiIcon Include="Resources\appicon.svg"
          ForegroundFile="Resources\Icons\appicon.svg"
          Color="#512BD4" />

<!-- Splash Screen -->
<MauiSplashScreen Include="Resources\Icons\appicon.svg"
                  Color="#512BD4"
                  BaseSize="128,128" />
```

Tyto výhody lze vidět na výstupech aplikace, kdy na Windows platformě, tak i na Android platformě vypadá aplikace téměř stejně a obsahuje stejné elementy.

Jedna z největších nevýhod je určitě použití některých NuGet balíčků, které může programátor potřebovat. Tím, že je .NET MAUI přece jen trochu jiné jak Xamarin.Forms, nemusí některé balíčky mít verzi právě pro .NET MAUI a vývoj pak bude muset počkat na spravení kompatibility nebo použít alternativu.

.NET MAUI je tedy velmi zajímavá technologie, která dokáže řešit čas docela elegantním způsobem. Pouze malá část vývoje aplikace je zaměřena na každou platformu zvlášť, protože většina kódu je opravdu sdílená a nevyžaduje specifický přístup exkluzivně ke každé platformě.

ZÁVĚR

Po průzkumu trhu je patrné, že převažují technologie zaměřené především na vývoj spíše mobilních aplikací a na desktopy je počet technologií o dost nižší. Proto je .NET MAUI v tomto unikátní a přinese na trh určitě nové možnosti, jako je přenesení již existujících aplikací (weby) na nativní klienty nebo z mobilních aplikací udělat i aplikaci pro desktop. I přes to, že je technologie v Preview verzi, vývoj šel docela hladce, i když se občas zasekl na čekání na opravu bugů nebo dodání některé funkcionality, které by za normálních okolností fungovaly.

Samotná aplikace je v současné fázi funkční a použitelná v praxi. Její vlastnosti dokážou pomoci investorovi s výběrem nemovitosti, a to tím, že je schopná přednést hodnoty důležité k rozhodnutí při nákupu zaznamenané nemovitosti. Pokud si investor není některými hodnotami jist, může si hodnoty pozměnit díky uložení záznamů do paměti zařízení. Díky tomu může editovat i starší záznamy a například zaktualizovat některé hodnoty (zvýšení nájmu nebo nákladů). To mu pomůže udržovat hodnoty aktuální v průběhu času držení nemovitosti.

Aplikace určitě není dokonalá a dala by se rozšířit například o pouze pracovní záznamy a záznamy již zakoupených nemovitostí s možností například uložení nájemní smlouvy, záznam kontaktu s nájemníkem nebo posílání zpráv nájemníkovi přímo z aplikace. To by ovšem znamenalo i úpravu některých bezpečnostních prvků, jako je například komunikace přes Internet nebo autorizace přihlášeného uživatele. Mohli by být přidány některé detailní propočty anebo rozpis nákladů na jednotlivé položky, jež pomůžou investorovi se zapsáním některých položek, které by si nemusel hned uvědomit, a tím zpřesnit výsledky zobrazené v detailu. Další potenciální rozšíření je určitě export záznamů nebo sdílená databáze, protože je aplikace multiplatformní jak na mobilní, tak i na desktopové zařízení. Tím by záznamy byly propojeny napříč těmito platformami a uživatel by tak měl přístup ke všem záznamům z jakéhokoliv zařízení.

SEZNAM POUŽITÉ LITERATURY

[1] TECHTARGET CONTRIBUTOR. Write once, run anywhere (WORA). WhatIs.com [online]. c1999 - 2022, March 2013 [cit. 2022-02-03]. Dostupné z: https://whatis.techtarget.com/definition/write-once-run-anywhere-WORA

[2] SHELDON, Amyra. 11 Popular Cross-Platform Tools for App Development (Updated Version 2021). Hackernoon.com [online]. 2016, March 20th 2020 [cit. 2021-10-15]. Dostupné z: https://hackernoon.com/9-popular-cross-platform-tools-for-app-development-in-2019-53765004761b

[3] Permissions on Android. *Android for Developers* [online]. 2005, 2022-01-27 [cit. 2022-01-27]. Dostupné z: https://developer.android.com/guide/topics/permissions/overview

[4] KOĎOUSKOVÁ, Barbora. CO JSOU PROGRESIVNÍ WEBOVÉ APLIKACE (PWA) A JAKÉ MAJÍ VÝHODY: CO JSOU PROGRESIVNÍ WEBOVÉ APLIKACE? *Rascasone: ČLÁNKY A KNOW-HOW* [online]. c2022, 18.10.2021 [cit. 2022-02-05]. Dostupné z: https://www.rascasone.com/cs/blog/progresivni-webova-aplikace-vyhody

[5] KOĎOUSKOVÁ, Barbora. CO JSOU PROGRESIVNÍ WEBOVÉ APLIKACE (PWA) A JAKÉ MAJÍ VÝHODY: NATIVNÍ MOBILNÍ APLIKACE VS. PROGRESIVNÍ WEBOVÉ APLIKACE. *Rascasone: ČLÁNKY A KNOW-HOW* [online]. c2022, 18.10.2021 [cit. 2022-02-05]. Dostupné z: https://www.rascasone.com/cs/blog/progresivni-webova-aplikace-vyhody

[6] LATHAM, Luke, Rick ANDERSON a Scott ADDIE. Build Progressive Web Applications with ASP.NET Core Blazor WebAssembly. *Microsoft: Technical documentation* [online]. c2022, 01/14/2022 [cit. 2022-02-06]. Dostupné z: https://docs.microsoft.com/cs-cz/aspnet/core/blazor/progressive-web-app?view=aspnetcore-6.0&tabs=visual-studio

[7] HYBRID APPS: AN OVERVIEW OF ADVANTAGES, LIMITATIONS & CONSEQUENCES FOR YOUR TESTING PHASES. *StarDust* [online]. c2022 [cit. 2022-02-06]. Dostupné z: https://www2.stardust-testing.com/en/blog-en/hybrid-apps

[8] Hybrid application (hybrid app). *WhatIs.com* [online]. c1999 - 2022, September 2019 [cit. 2022-02-03]. Dostupné z: https://searchsoftwarequality.techtarget.com/definition/hybrid-application-hybrid-app

[9] *Ionic Framework* [online]. c2020 [cit. 2022-02-06]. Dostupné z: https://ionicframework.com/

[10] *Electron* [online]. c2021 [cit. 2022-02-08]. Dostupné z: https://www.electronjs.org/

[11] Electron Docs: Get Started. *Electron* [online]. c2021 [cit. 2022-02-08]. Dostupné z: https://www.electronjs.org/docs/latest

[12] Michael. Electron: 1. část. *Kutáč* [online]. Ostrava - Poruba: Kutáč, c2014 - 2022, 01.02.2019 [cit. 2022-02-16]. Dostupné z: https://www.kutac.cz/weby-a-vse-okolo/electron-1-cast

[13] RAMEL, David. With .NET MAUI Delayed, Xamarin.Forms Remains Mobile Dev Option in .NET 6. *Visual Studio Magazine* [online]. c1996-2022, 11/09/2021 [cit. 2022-02-05]. Dostupné z: https://visualstudiomagazine.com/articles/2021/11/09/xamarin-maui.aspx

[14] Cross-platform mobile frameworks used... *Statista* [online]. July 2021 [cit. 2022-02-16]. Dostupné z: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/

[15] PHAM, LE. WHAT IS NATIVE APP?. *Magenest* [online]. c2021, November 30, 2021 [cit. 2022-02-16]. Dostupné z: https://magenest.com/en/native-app/

[16] The Majority of Americans' Mobile Time Spent... *Insider intelligence* [online]. Jul 9, 2020 [cit. 2022-02-19]. Dostupné z: https://www.emarketer.com/content/the-majority-of-americans-mobile-time-spent-takes-place-in-apps

[17] Number of smartphone subscriptions... *Statista* [online]. February 2022 [cit. 2022-02-19]. Dostupné z: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[18] CHUNG, Eddy. What is Xcode and why do I need it? *Zero To App Store* [online]. [cit. 2022-02-23]. Dostupné z: https://www.zerotoappstore.com/what-is-xcode-and-why-do-i-need-it.html

[19] Native vs Hybrid vs Cross-Platform – What To Choose and when?. *ETraverse* [online]. c2021 [cit. 2022-03-02]. Dostupné z: https://etraverse.com/blog/native-vs-hybrid-vs-cross-platform-what-to-choose-and-when/

[20] Hybrid Applications. OLSON, Scott, John HUNTER, Ben HORGEN a Kenny GOERS. *Professional Cross-Platform Mobile Development in C#.* Indianapolis: John Wiley, c2012, s. 297. ISBN 978-1-118-15770-1.

[21] ROSS, Bill. What to consider when architecting for mobile application development. *Equinox IT* [online]. 07/02/2014 [cit. 2022-03-02]. Dostupné z: https://www.equinox.co.nz/blog/what-to-consider-when-architecting-for-mobile-application-development

[22] WARDYNSKI, DJ. The 7 Best Cross-Platform Mobile Development Tools. *Brainspire* [online]. c2022, November 12, 2020 [cit. 2022-03-04]. Dostupné z: https://www.brainspire.com/blog/the-7-best-cross-platform-mobile-development-tools

[23] What is Xamarin?. *Microsoft Docs* [online]. c2022, 12/16/2021 [cit. 2022-03-04]. Dostupné z: https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin

[24] Chapter 1. What Is React Native?. *O'Reilly* [online]. c2022 [cit. 2022-03-05]. Dostupné z: https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch01.html

[25] THOMAS, Gaël. What is Flutter and Why You Should Learn it in 2020. *FreeCodeCamp* [online]. DECEMBER 12, 2019 [cit. 2022-03-05]. Dostupné z: https://www.freecodecamp.org/news/what-is-flutter-and-why-you-should-learn-it-in-2020/

[26] Accessing User Data and Resources. *Apple Developer* [online]. © 2022 [cit. 2022-03-07]. Dostupné z: https://developer.apple.com/design/human-interface-guidelines/ios/app-architecture/accessing-user-data/

[27] What is Multi-Factor Authentication (MFA) and How Does it Work. *OneLogin* [online]. c2022 [cit. 2022-03-10]. Dostupné z: https://www.onelogin.com/learn/what-is-mfa

[28] 12 Ways to Safeguard Mobile Apps from Menacing Cyber Attacks. *Skylark* [online]. c2022, 11th January 2021 [cit. 2022-03-10]. Dostupné z: https://www.skylark.com.sg/blog/12-ways-to-safeguard-mobile-apps-from-menacing-cyber-attacks/

[29] PANCHAL, Jaykishan. *Top 10 Mobile App Security Best Practices for Developers. Tripwire* [online]. c2022, FEB 14, 2018 [cit. 2022-03-10]. Dostupné z:

https://www.tripwire.com/state-of-security/security-awareness/top-mobile-app-security-best-practices-developers/

[30] HRANICKÝ, Jan. Lekce 2 - Technika útoku SQL injection. *Itnetwork.cz* [online]. c2022 [cit. 2022-04-26]. Dostupné z: https://www.itnetwork.cz/php/bezpecnost/technika-utoku-sql-injection

[31] HRANICKÝ, Jan. Lekce 3 - Jak se bránit proti SQL injection. *Itnetwork.cz* [online]. c2022 [cit. 2022-04-26]. Dostupné z: https://www.itnetwork.cz/php/bezpecnost/jak-se-branit-proti-sql-injection

[32] LIANG, Mia. Understanding Object-Relational Mapping: Pros, Cons, and Types. *AltexSoft* [online]. 11 Mar, 2021 [cit. 2022-04-26]. Dostupné z: https://www.altexsoft.com/blog/object-relational-mapping/

[33] SKINNER, Jeremy. FluentValidation. *FluentValidation* [online]. c2009-2021 [cit. 2022-04-27]. Dostupné z: https://docs.fluentvalidation.net/en/latest/index.html

[34] TOMEK, Pavel. Online kurz: 2. díl – Jak si spočítat výhodnost investice. *ADOL Monitor* [online]. c2022 [cit. 2022-04-29]. Dostupné z: https://www.adol.cz/blog-online-kurz-2-dil-jak-si-spocitat-vyhodnost-investice/

[35] HERMES, Dan. *Building Xamarin.Forms Mobile Apps Using XAML: Mobile Cross-Platform XAML and Xamarin.Forms Fundamentals*. California: Apress, 2019. ISBN 978-1-4842-4029-8.

[36] HERMES, Dan. *Xamarin Mobile Application Development: Cross-Platform C# and Xamarin.Forms Fundamentals*. New York: Apress, c2015. ISBN 978-1-4842-0215-9.

[37] Meet Android Studio. *Android Developers* [online]. Google Developers, 2022-04-07 [cit. 2022-05-04]. Dostupné z: https://developer.android.com/studio/intro

[38] .NET MAUI Slider Overview. *Syncfusion* [online]. c2001 - 2022, 23 Dec 2021 [cit. 2022-05-10]. Dostupné z: https://help.syncfusion.com/maui/slider/overview

[39] .NET MAUI Chart Overview. *Syncfusion* [online]. c2001 - 2022, 29 Sep 2021 [cit. 2022-05-10]. Dostupné z: https://help.syncfusion.com/maui/cartesian-charts/overview

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

MAUI	Multi-platform App UI.
WORA, WORE	Write once, run anywhere (everywhere).
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
API	Application Programming Interface
HTTPS	HyperText Transfer Protocol Secure
SDK	Software Development Toolkit
UX	User eXperience
UI	User Interface
GUI	Graphic User Interface
IDE	Integrated Development Rnvironment
XML	Extensible Markup Language
XAML	Extensible Application Markup Language
OEM	Original Equipment Manufacturer
RASP	Runtime Application Self-Protection
MFA	Multi-Factor Authentication
2FA	Twofactor Authentication
PBAC	Policy-Based Access Control
RBAC	Role-Based Access Control
CBAC	Content-Based Access Control
MVVM	Model-View-ViewModel
SQL	Structured Query Language
ORM	Object-Relational Mapping
OOP	Object-Oriented Programing

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH