Vývoj mobilních aplikací s pomocí frameworku .NET

Lukáš Pevný

Bakalářská práce
2021

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

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

Akademický rok: 2020/2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Lukáš Pevný**
Osobní číslo:	**A18070**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Vývoj mobilních aplikací s pomocí frameworku .NET**
Téma práce anglicky:	**The Development of Mobile Applications using the .NET framework**

Zásady pro vypracování

1. Popište současný stav technologií mobilního vývoje.
2. Zaměřte se na framework Xamarin Forms a jeho možného nástupce MAUI.
3. Zpracujte přehled doporučených návrhových vzorů, technik a architektur.
4. Navrhněte ukázkovou aplikaci s využitím frameworku .NET.
5. Vytvořte vzorová řešení demonstrující klíčové prvky navržené aplikace.
6. Demonstrujte výsledky.

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

Seznam doporučené literatury:

1. HERMES, Dan. Building xamarin.forms mobile apps using xaml: mobile cross-platform xaml and xamarin.forms fundamentals. New York, NY: Springer Science Business Media, 2019. ISBN 978-148-4240-298.
2. BENNETT, Jim. Xamarin in action: creating native cross-platform mobile apps. Shelter Island: Manning, 2018. ISBN 978-1617294389.
3. Xamarin.Forms – Xamarin | Microsoft Docs. [online]. Dostupné z: https://docs.microsoft.com/cs-cz/xamarin/xamarin-forms/
4. MVVM pattern | Microsoft Docs [online]. Dostupné z: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/gg405484(v=pandp.40)
5. Android Developer [online]. Dostupné z: https://developer.android.com/

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

Datum zadání bakalářské práce: **15. ledna 2021**
Termín odevzdání bakalářské práce: **17. května 2021**

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

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

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

Ve Zlíně dne 15. ledna 2021

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

Lukáš Pevný, v. r.
podpis diplomanta

ABSTRAKT

Cieľ bakalárskej práce je aplikácia, pomocou ktorej demonštrujem jej kľúčové prvky pomocou frameworku Xamarin.Forms. Práca sa skladá z teoretickej a praktickej časti. Čitateľ sa v teoretickej časti oboznámi s operačnými systémami, na ktoré je aplikácia vyvíjaná, o programoch, ktoré som použil pri vývoji a službách, vďaka ktorým sú spravené dôležité prvky aplikácie. V praktickej časti by sa mal čitateľ zoznámiť, ako jednotlivé prvky aplikácie fungujú s pohľadu užívateľa (ako by mal užívateľ s aplikáciou interagovať) a ako sú implementované s pohľadu systému (ako aplikácia spracuje požiadavku užívateľa).

Klíčová slova:

Xamarin.Forms, aplikácia na správu dlhov, mobilná aplikácia, DynamoDB, C#

ABSTRACT

Goal of the bachelor thesis is application, through which I demonstrate key elements with framework Xamarin.Forms. Bachelor thesis consists of theoretical and practical part. In theoretical part, the reader gets acquianted with operating systems on which the app is developer, about programs which I used in development and services that cover important elements of the application. In practical part reader gets acquianted how individual elements work from user point of view (how user should interact with the application) and from system point of view (how user requests are managed).

Keywords:

Xamarin.Forms, debt management application, mobile application, DynamoDB, C#

Rád by som sa poďakoval **Ing. et Ing. Erik Král, Ph.D.** za cenné rady a pomoc pri zhotovovaní tejto bakalárskej práce.

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

OBSAH

ÚVOD

Motivačným prvkom tejto práce bolo vytvorenie lepšieho systému pre prehľadnejšiu evidenciu osobných dlhov. Nenašiel som existujúcu aplikáciu alebo web stránku, ktorá by vyhovovala mojim požiadavkám. Dôvod pre vytvorenie mobilnej aplikácie bol jednoduchý. Dosiahnuť to, aby mal užívateľ stále svoj zoznam dlhov pri sebe iba s potrebou internetového pripojenia. Cieľ bol spočiatku prostý, vytvoriť aplikáciu, kde si budú môcť užívatelia pozerať dlhy, ktoré niekomu dlhujú. Vzhľadom na možnosti, ktoré inteligentné telefóny predstavujú sa počas vývoja veľa funkcií pridalo, ktoré na začiatku vôbec neboli v pláne.

V teoretickej časti sú zhrnuté informácie o možnostiach vývoja mobilných aplikácií, prehľad operačných systémov iOS a Android a ich architektúry. Ďalej sa v teoretickej časti nachádzajú informácie o použitých programoch a knižniciach v projekte a dôvod ich výberu. V poslednej časti sú popísané návrhové vzory, techniky a architektúry, ktoré sú doporučené a ktoré sú použité na prípady užitia, na ktoré sú vhodné.

V praktickej časti sa nachádza návrh aplikácie a konkrétne požiadavky, podľa ktorých bola aplikácia vyvíjaná. Ďalej sa v tejto časti vyskytuje kapitola o kľúčových prvkoch aplikácie, kde je popísané, ako tieto prvky boli naprogramované a akú funkciu zastávajú v aplikácií. Na uzatvorenie tejto časti je v práci sa kapitola, ktorá demonštruje celú aplikáciu a ukazuje, ako by mal užívateľ s aplikáciou interagovať a čo má očakávať po interakcií s konkrétnym prvkom v aplikácií.

Táto bakalárska práca nadväzuje na seminárnu prácu [1] z predmetu A4TDP.

I. TEORETICKÁ ČÁST

1 VÝVOJ MOBILNÝCH APLIKÁCIÍ

Vývoj mobilných aplikácií je proces vytvárania aplikácie, ktorá je určená pre spustenie na mobilných zariadeniach. Podľa príspevku [2], typické mobilné aplikácie využívajú internetové pripojenie na prácu so zdrojmi, ktoré nie sú uložené lokálne na zariadení (napr. databáza, zložité a náročné operácie pre mobilný telefón, komunikáciu s API).

V moderných mobilných telefónoch (smartfónoch) sú dominantné 2 operačné systémy: iOS a Android. Hlavný rozdiel medzi týmito operačnými systémami je ten, že iOS je výlučne používaný firmou Apple iba na svojich zariadeniach. Operačný systém Android je pre výrobcov inteligentných telefónov dostupný pre používanie za splnenie určitých požiadaviek, jedno z nich je predinštalované Google aplikácie. Viac o týchto operačných systémoch je popísané v kapitolách 5 a 6.

Existuje niekoľko vývojárskych prístupov pri vývoji mobilných aplikácií. To sú nasledovné:

- Natívne mobilné aplikácie
- Cross-platform natívne mobilné aplikácie
- Hybridné mobilné aplikácie
- Progresívne web aplikácie

Dodatočné informácie ku všetkým typom vývojárskych prístupov je možné získať z príspevku [2].

1.1 Natívne mobilné aplikácie

Natívne mobilné aplikácie sú písané v programovacích jazykoch, ktoré sú podporované priamo vlastníkmi operačného systému. Ako je zrejmé z tejto stránky [3], pre operačný systém iOS to je Swift. Pre operačný systém Android sú to hlavne jazyky Java a Kotlin podľa dokumentácie [4].

Výhody

- priamy prístup k API zariadenia
- najlepší beh programu

Nevýhody

- vyššia cena pri vytváraní a podporovaní aplikácie
- dva rôzne zdrojové kódy pre každú platformu

1.2 Cross-platform mobilné aplikácie

Cross-platform aplikácie môžu byť napísané v niekoľkých programovacích jazykoch alebo frameworkoch. Aplikácia je následne zo zdieľaného kódu prekladaná do natívnej aplikácie bežiacej na konkrétnom operačnom systéme.

Výhody

- jeden zdrojový kód pre viacero platforiem
- jednoduchšia podpora aplikácie

Nevýhody

- závislosť na knižniciach, ktoré sú preložené z natívnych funkcií zariadenia
- možné limitácie výkonu zariadenia z dôvodu prekladania kódu do natívneho

Z prieskumu portálu stackoverflow [5], ohľadom používaných frameworkov vývojármi je jasné, že najpoužívanejší framework na cross-platform aplikácie je Flutter s 68,8 %. Spomedzi cross-platform frameworkov sa v danom prieskume nachádzajú ešte React Native s 57,9 % a Xamarin s 45,4 %. Percentá v tomto prieskume znamenajú, koľko percent opýtaných pracujú s daným frameworkom prejavili záujem v pokračovaní práce s daným frameworkom.

1.3 Hybridné mobilné aplikácie

Hybridné mobilné aplikácie sú vytvárané pomocou štandardov tvorenia webu – JavaScript, CSS, HTML, atď. Následne sú zabalené ako aplikácia s inštaláciou. Na rozdiel od natívnych aplikácií, tieto aplikácie pracujú s „web kontajnerom“. Pri spustení aplikácie je dané okno otvorené cez celú obrazovku a aplikácia sa správa normálne ako natívna. Tento kontajner taktiež zabezpečuje premostenie medzi aplikáciou a API zariadenia.

Výhody

- zdieľa zdrojový kód medzi webom a mobilnou aplikáciou
- používanie zručností z tvorenia webu pre vytvorenie mobilnej aplikácie

Nevýhody

- nižší výkon v porovnaní s natívnymi aplikáciami
- limitovaná podpora pre funkcie natívneho zariadenia

Z prieskumu portálu stackoverflow [5], ohľadom používaných frameworkov vývojármi sa tu nachádza iba framework Cordova s 28,7 %.

1.4 Progresívna webová aplikácia

Progresívna webová aplikácia prináša alternatívny prístup k vývoji mobilných aplikácií. PWA aplikácie sa nenachádzajú v obchode s aplikáciami a ani nie je potrebná inštalácia. PWA aplikácia si užívateľ iba „pripne“ na plochu mobilného telefónu. V podstate to je aplikácia, ktorá je stále web stránka, bez lišty a správa sa ako normálna aplikácia. PWA je web aplikácia, ktorá využíva funkcie prehliadača.

Výhody

- tá istá aplikácia k dispozícií ako web a mobilná aplikácia
- žiadna inštalácia nie je potrebná, k dispozícií cez URL adresu

Nevýhody

- limitovaná podpora pre funkcie natívneho zariadenia
- schopnosti aplikácie záležia od používaného prehliadača

2 XAMARIN FORMS

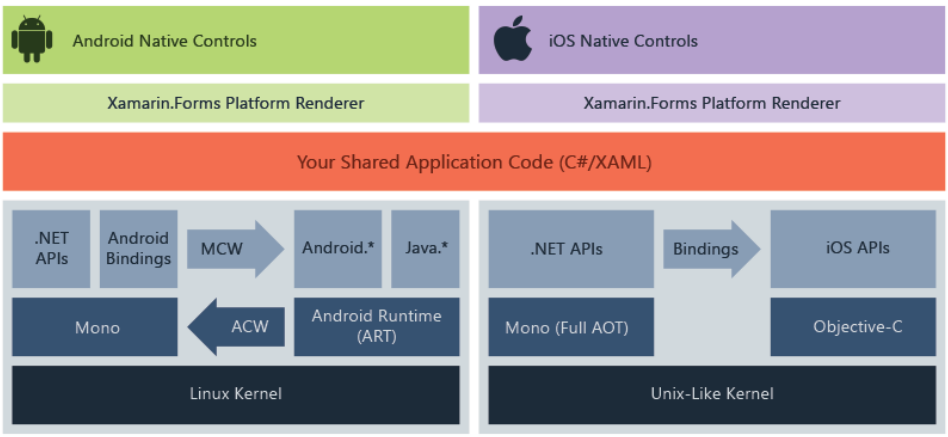
Xamarin.Forms je open source[1] user interface framework[2]. Podľa dokumentácie [6], Xamarin.Forms umožňuje vývojárom vytvárať Xamarin.Android, Xamarin.iOS a Windows aplikácie z jedného zdieľaného zdrojového kódu. Xamarin.Forms umožňuje vytvárať užívateľské rozhrania pomocou XAML s kódom na pozadí v jazyku C#. Tieto rozhrania sú renderované do natívnych ovládacích prvkov pre jednotlivú platformu.

Za behu, Xamarin.Forms renderuje cross-platform elementy užívateľského rozhrania na natívne elementy pre Xamarin.Android, Xamarin.iOS a UWP. Xamarin.Forms aplikácia typicky pozostáva zo zdieľanej .NET Standard library a individuálnych projektov pre platformy. Zdieľaná platforma pozostáva z XAML alebo C# okien a ostatnej business logiky. Projekty pre dané platformy môžu obsahovať dodatočný zdrojový kód pre danú platformu. Podrobnejšie ukázanie vrstiev Xamarin.Forms je na obrázku 1.

Xamarin.Forms je určený pre vývojárov dosiahnuť tieto ciele:

- zdieľať užívateľské rozhranie naprieč platformami
- zdieľať kód a logiku naprieč platformami
- písať cross-platform aplikácie v jazyku C# vo vývojárskom prostredí Visual Studio

Viac o tomto frameworku je taktiež možné sa dočitať v knihe [7].



Obrázek 1 Xamarin.Forms – vrstvy [6]

[1] Softvér, ktorý môže ktokoľvek prezerať, upravovať a vylepšovať https://opensource.com/resources/what-open-source

[2] Softwarová štruktúra slúžiaca ako pomoc pri programovaní

3 MAUI

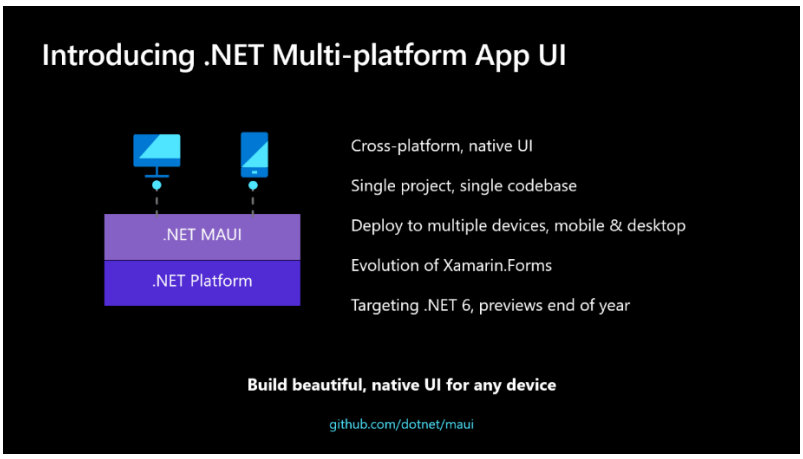
.NET MAUI je evolúciou frameworku Xamarin.Forms. Podľa príspevku [8] na blogu .NET, .NET MAUI zjednodušuje možnosti vývojárov .NET a poskytne framework, ktorý podporuje všetky moderné platformy: Android, iOS, macOS a Windows. Introdukciu ku frameworku .NET MAUI a hlavné výhody je možné vidieť na obrázku 2.

Hlavné vylepšenia .NET MAUI oproti Xamarin.Forms podľa príspevku [9] na blogu Xamarin, sú:

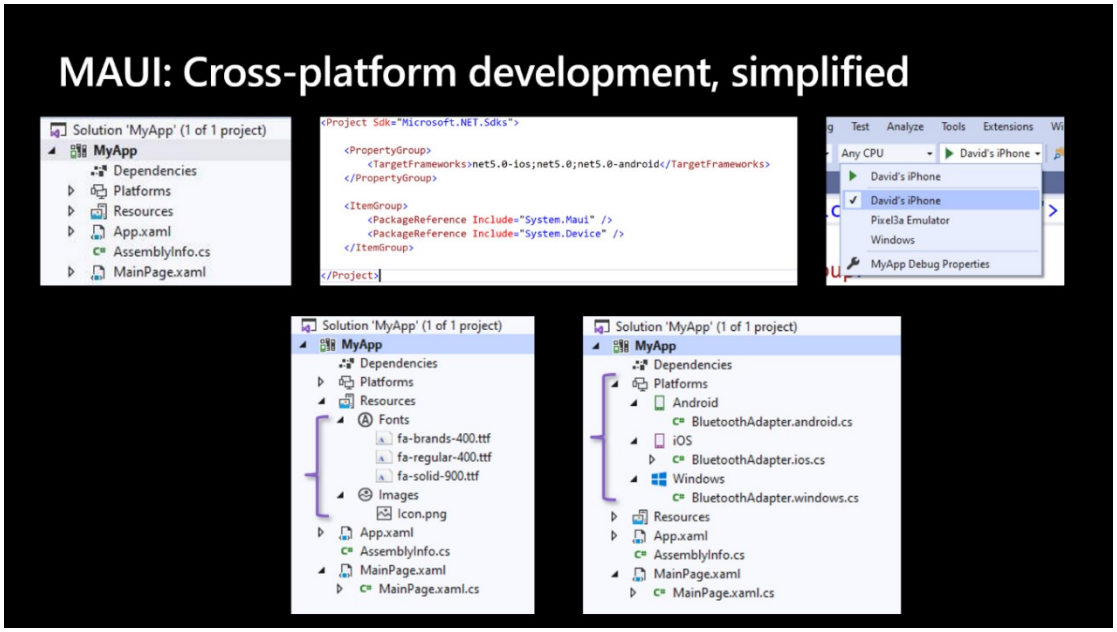
- ešte viac zjednodušené a vylepšené zdieľanie kódu medzi platformami
- moderné architektúry
- výkon
- dizajn (Fluent API a Material Design)
- .NET Hot Reload

3.1 Vylepšené zdieľanie kódu

Podľa príspevku [8] na blogu .NET, .NET MAUI zjednodušuje štruktúru kódu na 1 projekt, ktorý bude možné preložiť na viacero platforiem. Toto znamená, že pri preklade si vývojár jednoducho zvolí na aké zariadenie chce kód preložiť. Taktiež bude umožnené všetky obrázky a fonty použité v aplikácií mať na jednom mieste a MAUI sa automaticky postará o nastavenie na konkrétnej platforme. Prístup k špecifickým kódom pre konkrétnu platformu je tiež zjednodušený. Kód sa vkladá do „Platforms“ do príslušnej časti pre daný operačný systém. Ako táto štruktúra kódu vyzerá priamo v programovacom nástroji Visual Studio je možné vidieť na obrázku 3.



Obrázek 2 .NET MAUI – introdukcia [8]



Obrázek 3 .NET MAUI – štruktúra kódu [8]

3.2 Moderné architektúry

.NET MAUI bude naďalej podporovať Model-view-viewmodel (10.1). .NET MAUI bude podporovať aj Model-View-Update. Použitie MVU v .NET MAUI by malo vyzerať ako na obrázku 4. Počas písania bakalárskej práce viac informácií o použití MVU v .NET MAUI dostupných nebolo.

```
readonly State<int> count = 0;

[Body]
View body() => new StackLayout
{
    new Label("Welcome to .NET MAUI!"),
    new Button(
        () => $"You clicked {count} times.",
        () => count.Value ++)
    )
};
```

Obrázek 4 MVU – ukážka [8]

3.3 .NET Hot Reload

Podľa príspevku [9] na blogu Xamarin, hot reload je funkcia, ktorá umožňuje automatické nahranie zmeneného kódu do zariadenia bez nutnosti znovu inštalácie aplikácie. Dokonca

nie je potreba aplikácia ani vypínať, všetko je prevedené v reálnom čase. Pre XAML je táto funkcia už dostupná a pripravuje sa aj pre zmenu kódu v C#.

3.4 Podpora Xamarin.Forms

Aktuálna verzia Xamarin.Forms je 5.0.0. Podľa informácií z repozitáru [10], táto verzia bude podporovaná do novembra 2022. Ďalšie vydanie hlavnej verzie bude už pod .NET MAUI. Existujúce a nové Xamarin.Forms aplikácie budú migrovať na .NET MAUI s pomocou nástrojov a dokumentácie. Tieto nástroje budú poskytnuté v rámci časovej ukážky .NET 6 (začiatok 2021 – november 2021).

3.5 Dostupnosť .NET MAUI

Podľa informácií z repozitáru [11], stabilná verzia .NET MAUI má byť vydaná v novembri 2021.

4 EXTENSIBLE APPLICATION MARKUP LANGUAGE (XAML)

4.1 Všeobecné informácie

eXtensible Application Markup Language (ďalej aj XAML) je jazyk postavený na jazyku XML. Podľa dokumentácie [13], XAML je vyvíjaný spoločnosťou Microsoft a je používaný hlavne pre vytváranie užívateľských rozhraní v službách Windows Presentation Foundation (WPF), Silverlight, Windows Runtime, Universal Windows Platform (WPF) a Xamarin.Forms.

XAML dovoľuje vývojárom vytvárať užívateľské rozhrania pomocou tzv. „markup[3]“ jazyka radšej než pomocou kódu. XAML je tiež vhodný na použitie aplikačnej architektúry MVVM (10.1). XAML definuje užívateľské rozhranie, ktoré je pomocou XAML prepojení prepojené na príslušný ViewModel.

Výhody použitia XAML oproti ekvivalentnému C# kódu

- XAML je väčšinou čitateľne vyhovujúcejší
- Architektúra dieťa – rodič je lepšie viditeľná v XAML
- XAML môže byť vytvorený programátormi ako aj jednoducho generovaný nástrojmi na dizajn

Nevýhody použitia XAML, hlavne kvôli limitácií markup jazykov

- XAML nemôže obsahovať kód, čiže všetky obsluhy udalostí musia byť v separátnom súbore
- XAML nemôže obsahovať slučky (ListView može generovať elementy)
- XAML nemôže volať classy, ktoré nemajú konštruktor bez parametrov
- XAML nemôže volať metódy

Rozdiel medzi XAML a XML je v tom, že XAML obsahuje unikátny syntax. Najhlavnejšie sú „property“ elementy, pripojené vlastnosti (attached properties) a rozšírenia markup.

Viac informácií o jazyku XAML v spojení s Xamarin.Forms sú dostupné v knihe [14].

[3] markup – jazyk, ktorý používa tagy, ktoré definujú grafické elementy [12]

4.2 Ukážka XAML súboru v Xamarin.Forms

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="App1.Page1">
    <ContentPage.Content>
        <StackLayout>
            <Label Text="Welcome to Xamarin.Forms!"
                VerticalOptions="CenterAndExpand"
                HorizontalOptions="CenterAndExpand" />
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

Obrázek 5 Ukážka XAML súboru v Xamarin.Forms

Novovytvorená stránka v Xamarin.Forms vyzerá ako na obrázku 5 a obsahuje nasledovné elementy a vlastnosti elementov:

ContentPage – stránka zobrazujúca jedno okno, väčšinou StackLayout alebo ScrollView

xmlns, xmlns:x – referencie na URI vlastnené Xamarinom a Microsoftom, ich funkcia je identifikácia verzie

xmlns:local – deklarácia „namespace“ nám umožňuje prístup k ostatným triedam z projektu

x:Class – názov triedy tzv. „na pozadí“ daného XAML súboru

StackLayout – organizuje prvky vo vnútri StackLayoutu v jednosmernom liste, horizontálne alebo vertikálne

Label – element, na zobrazovanie jedno ale aj viac riadkového textu. Môžu byť nastavované fonty, veľkosti a rôzne úpravy.

Text – zobrazovaný text

VerticalOptions – definuje, ako sa má prvok zarovnať vertikálne v rámci StackLayoutu (v tomto prípade)

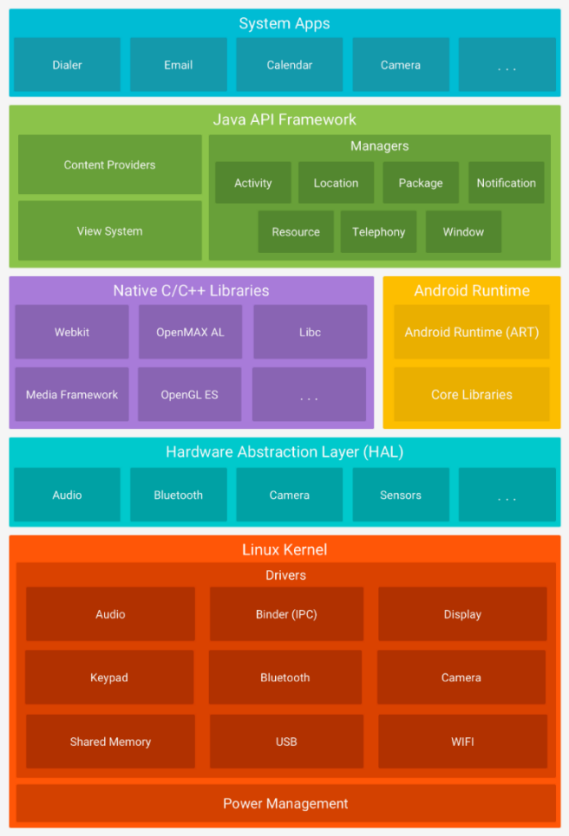
HorizontalOptions - definuje, ako sa má prvok zarovnať horizontálne v rámci StackLayoutu (v tomto prípade)

5 ANDROID

Android je operačný systém spoločnosti Google. Patrí k jedným z najrozšírenejších systémov podporujúcich zariadenia rôznych výrobcov. Podľa prieskumu [15] spoločnosti statcounter, má Android podiel na trhu 71.9 % ku dňu 22.3.2021 za posledných 12 mesiacov. Je založený na Linuxe. Na základe informácií z príspevku [16], výhodou programovania je to, že stačí programovať na jednu verziu androidu a na všetkých zariadeniach podporujúcich danú verziu by aplikácia mala fungovať. Android je z väčšej časti naprogramovaný v programovacom jazyku Java. Podľa článku [17], Android je napísaný z cca. 30% v jazykoch C,C++ a 8% má zastúpenie jazyk XML. Zdrojový kód Androidu spadá pod open-source licenciu. Niektoré funkcie Androidu: intuitívne grafické rozhranie, konektivita, podpora rôznych multimediálnych formátov, posielanie SMS správ, podpora internetového prehliadača, multi-tasking, atď.

5.1 Architektúra platformy

Android sa delí na vrstvy podľa práce. Od hardwaru až po systémové aplikácie. Delenie vrstiev je graficky znázornené na obrázku 6.



Obrázek 6 Architektúra systému Android [18]

Nasledujúce podkapitoly boli čerpané zo zdroja [18].

5.1.1 Linux Kernel

Linux kernel je základom operačného systému android. Vďaka správe kľúčového hardwaru dovoľuje výrobcom mobilných zariadení vyvíjať ovládače.

5.1.2 Hardware abstrakčná vrstva

Vďaka tejto vrstve môže mať Java API Framework prístup k niektorým hardwarovým prvkom. Je to z toho dôvodu, že táto vrstva pre dané komponenty obsahuje a implementuje knižnice na správu. Pri zavolaní Java API frameworkom na daný komponent spravovaný touto vrstvou, Android načíta modul z knižnice pre tento hardwarový komponent.

5.1.3 Android Runtime

Podľa príspevku [19] z diskusného fóra Stackoverflow, runtime je popísanie softwaru alebo inštrukcií, ktorý slúži na preklad inštrukcií, ktoré sme nenapísali explicitne, ale sú potrebné na správnu exekúciu kódu. V prípade Androidu to je Android Runtime (ART) a Dalvik (od verzie Androidu 5 úplne nahradený pomocou ART).

5.1.4 Natívne C/C++ knižnice

Natívne knižnice C/C++ sa v Androide nachádzajú z toho dôvodu, že veľa komponentov systému sú napísané v natívnom kóde, ktoré na exekúciu potrebujú natívne knižnice C/C++. Java API Framework poskytuje funkcie týchto knižníc aplikáciám.

5.1.5 Java API Framework

Všetky komponenty systému Android sú poskytované pomocou API vďaka tejto vrstve. Tieto API sú základným kameňom pri vytváraní Android aplikácií. Komponenty a služby poskytované touto vrstvou sú: View System (tvorba layoutov – to ako bude aplikácia vyzerať), Resource Manager (zabezpečuje prístup k zdrojom, ktoré sú používané v kóde napr. Obrázky, logá atď.), Notification Manager (správca notifikácií – možnosť vôbec nejakého doručenia a zobrazenia), Content Providers (spôsob prístupu k dátam vytvorených inou aplikáciou).

5.1.6 Systémové aplikácie

Android v stave, v akom sa nainštaluje, obsahuje niekoľko systémových aplikácií, ako SMS správy, kalendár, hodiny, internetový prehliadač atď. Tieto systémové aplikácie nemajú žiadne špeciálne postavenie v porovnaní s aplikáciami stiahnutými z tretích strán. Vo väčšine prípadov ale nemôžu byť odinštalované. Tieto aplikácie neslúžia len užívateľom ale môžu slúžiť aj programátorom na spojenie ich aplikácie s konkrétnou systémovou aplikáciou.

6 IOS

Tento operačný systém je vyvíjaný spoločnosťou Apple. Jeden z hlavných rozdielov iOSu a Androidu je ten, že iOS je exkluzívny len na zariadenia, ktoré sú vytvárané spoločnosťou Apple. Podľa prieskumu [15] spoločnosti statcounter, podiel používateľov s operačným systémom iOS je 27,33 % k dátumu 23.3.2020 za posledných 12 mesiacov. Podľa článku [20], ovládanie užívateľského rozhrania tohoto operačného systému je založené výhradne na gestách. Funkcie iOSu sú vo veľkej časti rovnaké ako Androidu. IOS nie je open source ako je Android. Vďaka tomuto ho firma Apple používa výhradne len pre svoje vlastné zariadenia.

Informácie, ku všetkým podkapitolám boli čerpané z článku [20].

6.1 Architektúra platformy

IOS architektúra sa taktiež delí do vrstiev podobne ako Android. Podrobnejšie si ich pozrieme s pohľadu od hardwaru až po aplikácie.

Obrázek 7 Architektúra systému iOS [20]

6.1.1 Core OS

Core OS je najnižšia vrstva operačného systému iOS. Táto vrstva pracuje s hardwarom zariadení, v ktorých sa tento operačný systém nachádza. Niektoré technológie založené na tejto vrstve sú: autentifikácia, 64-bit podpora, framework pre servisy bezpečnosti, Bluetooth framework. Core OS vrstva poskytuje tieto služby: štandardné vstupy a výstupy, networking, threading, služby bezpečnosti.

6.1.2 Core Services

Vrstva so základnými službami OS. Tieto služby sú :

- **Marco Address Book –** umožňuje programátorovi prístup k databáze kontaktov
- **Cloud Frame Kit –** zabezpečuje presun dát medzi aplikáciou a iCloudom
- **The framework of basic data –** technológia manažovania dátového modelu Model View Controlleru
- **Core Foundation framework –** interface zabezpečujúci manažovanie kritických dát a aplikačných službách
- **The Core Location framework –** informácie o polohe
- **Core frame motion –** zabezpečuje prístup ku všetkým dátam z akcelerometra
- **Foundation Framework**
- **Healthkit Framework –** framework na manažovanie informácií spojených so zdravím užívateľa
- **Homekit Framework –** framework zabezpečujúci komunikáciu a kontrolu nad pripojenými zariadeniami v dome
- **Social Framework –** framework zabezpečujúci prístup k účtom užívateľa
- **StoreKit frame –** podporuje nákup obsahu a služieb v aplikáciách (ASIN – App Purchase)

6.1.3 Media Layer

Vrstva pre grafiku, audio a video technológiu.

Graphics framework :

- **UIKit Graphics –** vyšší level podpory pre návrh obrázkov
- **Graphics Core Framework –** nástroj na kreslenie, podporuje aj vlastné 2D a renderuje na základe obrazových vektoroch
- **Core Animation –** technológia, optimalizujúca zážitok z animácií
- **Core Images –** manažuje obrázky a poskytuje podporu pre video a obrázky nedeštruktívnym spôsobom
- **OpenGL ES and GLKit –** manažuje 2D a pokročilú 3D zrýchlenie hardwarovým interfacom
- **Metal –** poskytuje veľmi vysoký výkon pre prácu s grafickým vykresľovaním

6.1.4 Cocoa Touch

Používateľom prístupný framework. Používaný na „zbuildenie“ softwaru a programov na spustenie v iOSe. Cocoa Touch obsahuje iné nástroje grafických elementov ako cocoa (macOS API). Nástroje aplikácií založených na Cocoa Touch sú obsiahnuté v iOS SDK a iOS platforme.

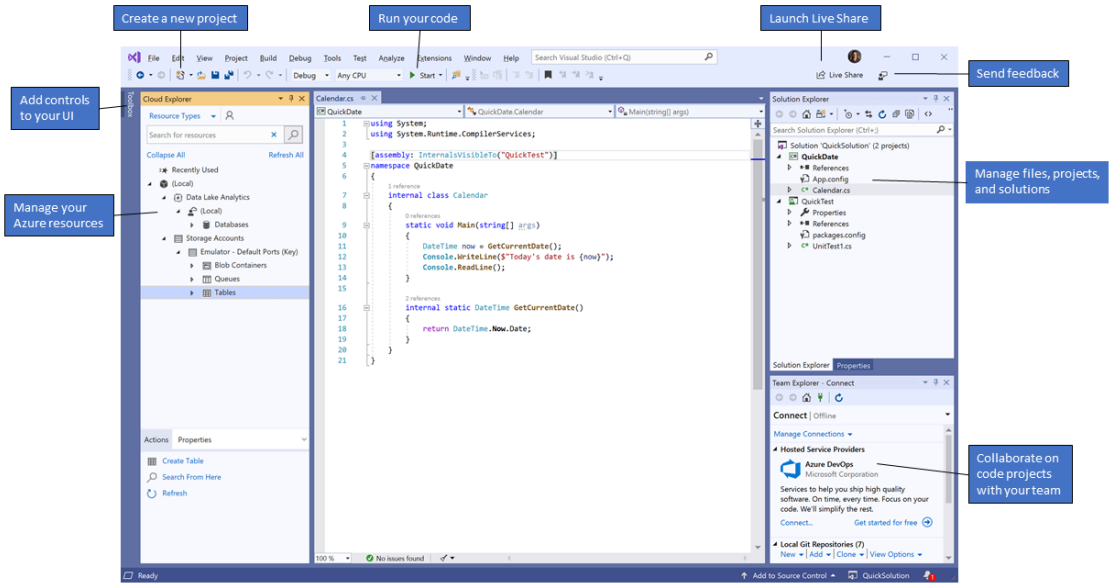
- **EventKit Frame** – poskytnutie driverom pohľad na štandardné systémové rozhrania
- **GameKit Marco** – implementuje podporu pre Herné Centrum. Dovoľuje užívateľom zdieľať herné informácie online
- **iAd Framework** – umožňuje poskytovať reklamy na základe
- **MapKit Marco** – displacement mapping (počítačová grafická technika)
- **Pushkin Framework** – podpora registrácií pre VoIP aplikácie
- **Marco Twitter** – podporuje generovanie „tweetov“ a podporuje vytváranie URL na prístup k Twitter službe
- **UIKit Framework** – zásadný význam pre implementáciu grafických aplikácií založených na iOS infraštruktúre. Jedny z najdôležitejších funkcií UIKitu sú : podpora multitaskingu (súbežné spracovanie úloh), manažovanie aplikácií a základná infraštruktúra, manažovanie užívateľského interfacu, podpora pre udalosť dotyku a pohybu

7 PROGRAMY

V tejto podkapitole sa nachádzajú informácie k programom, ktoré boli pri vývoji použité.

7.1 Vývojárske prostredie Visual Studio Community 2019

Podľa dokumentácie [22] k tomuto nástroju, je Visual Studio prostredie na úpravu, debugovanie[4], vytváranie a publikáciu aplikácií. Na výber je možnosť pracovať v rôznych programovacích jazykoch, ako C#, C++, JavaScript a mnohé ďalšie. Podobne ako pri programovacích jazykoch aj platforiem, na ktoré je možno programovať, je hneď niekoľko. Niektoré z možností sú: Android, iOS, Linux, Windows, atď. Taktiež je na výber z niekoľkých typov projektov ako: konzola, cloud, IoT, machine learning atď. Hlavné grafické rozhranie Visual Studia po vybraní jazyka, platformy a typu projektu vyzerá ako na obrázku 8.



Obrázek 8 Visual Studio Community 2019 – rozhranie [21]

Popis jednotlivých okien:

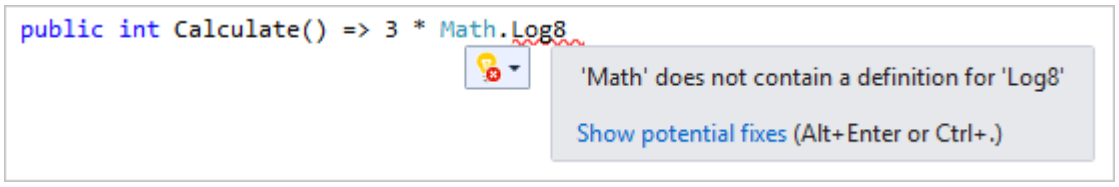
- **Solution explorer –** okno, v ktorom vidíme všetky súbory, ktoré sa nachádzajú v našom projekte
- **Okno editovania (v strede) –** slúži na programovanie

[4] debugovanie je spôsob, akým programátor hľadá miesto výskytu chyby, ktorú potrebuje opraviť

- **Team Explorer** – možnosť spolupráce na kóde medzi tímom ľudí

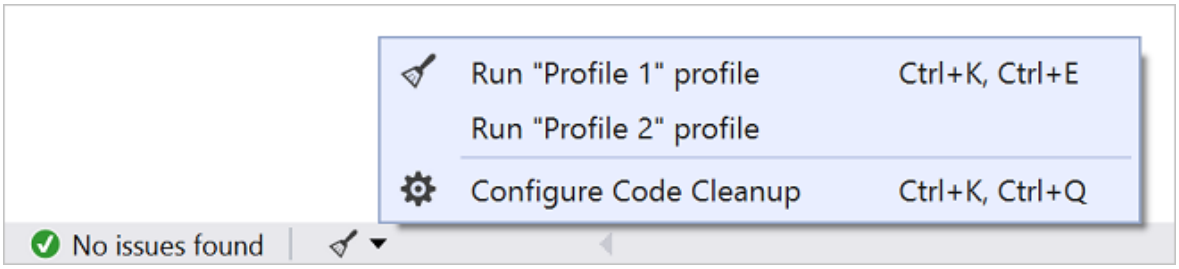
Niektoré z populárnych funkcií Visual Studia :

- **Rýchla akcia –** oprava alebo sugescia, čo môže programátor potrebovať



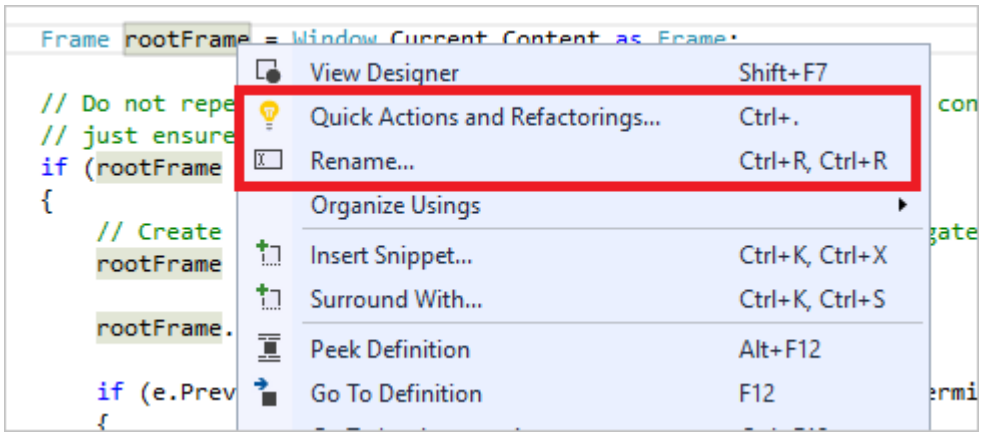
Obrázek 9 Rýchla akcia [22]

- **Vyčistenie kódu (len pre C#) –** na základe nastavení a analýz upraví formátovanie kódu



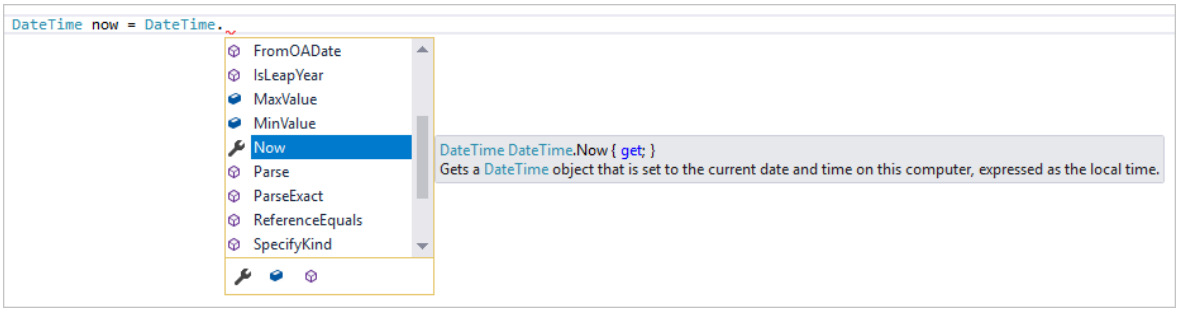
Obrázek 10 Vyčistenie kódu [22]

- **Refactoring –** operácie ako: inteligentné premenovanie premenných, zmena poradia parametrov, atď.



Obrázek 11 Refactoring [22]

- **IntelliSence –** okno, ktoré na základe kontextu poskytuje návrhy, čo môže programátor použiť



Obrázek 12 IntelliSence [22]

Dôvod použitia – Xamarin na operačnom systéme Windows je podporovaný jedine v tomto vývojárskom prostredí. V prípade, že by bol dostupný v nejakých iných, stále by som volil toto vývojárske prostredie vzhľadom na skúsenosti s týmto prostredím.

8 KNIŽNICE

V tejto podkapitole sa nachádzajú informácie o knižniciach, ktoré boli pri vývoji použité.

AWSSDK.DynamoDBv2

Táto knižnica zjednodušuje komunikáciu s databázou DynamoDB. Viac o tejto databáze je možné sa dočítať v tejto kapitole – 9.2.1. [23]

LottieXamarin a LottieAndroid

Táto knižnica slúži na vykreslenie animácií. Animácie musia byť vytvorené v software Adobe After Effects a vyexportované s pomocou rozšírenia Bodymovin do formátu json. Animácie, ktoré som použil sa nachádzajú priamo na stránke služby, ktorá túto knižnicu zastrešuje - https://lottiefiles.com/ [24]

firebaseNet

Táto knižnica zjednodušuje komunikáciu (odosielanie) push notifikácií[5], ktoré je zastrešované službou Firebase Cloud Messaging. [26]

Html Agility Pack

Táto knižnica parsuje[6] HTML stránky. Pomocou tejto knižnice si z nich môže vývojár vytiahnuť potrebné údaje a spracovať ich pre potrebné účely. [27]

MonkeyCache a MonkeyCache.FileStore

Táto knižnica slúži na jednoduché cachovanie[7] akýchkoľvek dát na špecifický čas v hocijakej .NET aplikácií. [28]

Plugin.FirebasePushNotification

Táto knižnica slúži na spracovanie prichádzajúcich push notifikácií. Využívam ju hlavne na získanie tokenu zariadenia, podľa ktorého sa notifikácie rozosielajú. [29]

[5] push notifikácia – notifikácia, ktorá sa objaví na mobilnom zariadení. Na prijatie nemusia byť používatelia v aplikácií ani nemusia používať svoje zariadenie. [25]
[6] parsuje – extrahuje dáta z HTML stránok
[7] cachovanie – ukladanie informácií ku ktorej procesor pristupuje veľmi rýchlo

Rg.Plugins.Popup

Rg.Plugins.Popup je knižnica pre Xamarin.Forms pomocou ktorej je možné vytvárať popup (vyskakovacie) stránky a taktiež animácie pre príchod a odchod týchto stránok. [30]

Xam.Plugins.Notifier

Jednoduchá knižnica, ktorá zabezpečuje vypisovanie push notifikácií lokálne. [31]

Xamarin.CommunityToolkit

Táto knižnica je kolekciou grafických prvkov a ďalších vecí, ktoré vývojári zvykli replikovať [32]. Táto knižnica by mala zastrešiť niektoré z bežných problémov, s ktorými sa vývojár stretne. Niektoré prvky boli presunuté z Xamarin.Forms do tejto knižnice z dôvodu vylepšovania a rýchlejšej dostupnosti vývojárom ako pri vydaniach Xamarin.Forms [33].

Xamarin.Essentials

Na rozdiel od knižnice Xamarin.CommunityToolkit, ktorá sa hlavne zameriava na grafické prvky sa Xamarin.Essentials zameriava na prácu so zariadením. Táto knižnica ponúka v zdieľanom kóde C# API pre základné informácie o zariadení, ktoré by boli nutné získavať na špecifickej platforme špecifickým kódom [34]. Niektoré z týchto informácií sú napríklad pripojenie k internetu, akcelerometer, kompas, kontakty, otvorenie prehliadača a ďalšie [35].

Xamarin.FFImageLoading a Xamarin.FFImageLoadingForms

Knižnica na jednoduché a rýchle načítanie obrázkov. S pomocou tejto knižnice je možné obrázky aj cachovať. [36]

Xamarin.Firebase.Config

Knižnica, ktorá sa zaoberá prepojením Firebase Remote Config a Androidu. [37]

Xamarin.Forms.PancakeView

Knižnica, ktorá sa rozširuje prvok ContentView[8] pomocou zaoblených rohov, okrajov, tieňov a mnohým ďalším. [38]

[8] ContentView - https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/layouts/contentview

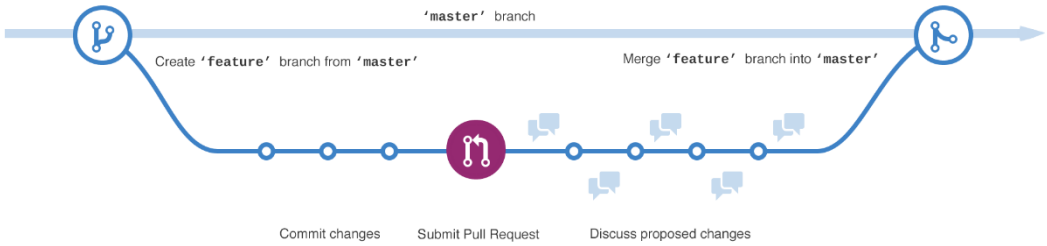
9 SLUŽBY TRETÍCH STRÁN

V tejto podkapitole sú priblížené služby tretích strán, ktoré boli pri vývoji použité ale aj tie, ktoré s aplikáciou komunikujú pri používaní užívateľom.

9.1 Github

Podľa príručky [39] Github je platforma, ktorá slúži na uloženie zdrojového kódu, kolaboráciu a dohľad nad verziami. Hlavné funkcie Githubu sú :

- **Repozitár** – repozitár je priečinok, ktorý zahŕňa všetko čo sa v projekte používa
- **Branching** – možnosť ako pracovať na rôznych verziách projektu v tom istom repozitári



Obrázek 13 Github – Branching [39]

Na obrázku 13 je ukázané vytvorenie „feature“ branchu. Ďalej je ukázané vytvorenie zmien, vytvorenie „pull requestu“, diskutovanie ohľadom zmien a následné zmergovanie (spojenie) „feature“ branchu do master.

- **Pull request –** podľa príspevku [40] na diskusnom fóre stackoverflow, je to jeden zo základných prvkov pri kolaborácií. Dovoľuje z vami vytvoreného branchu, publikovať „pull request“. Daný „pull request“ obsahuje informácie o vykonaných zmenách v porovnaní s master branchom. Potom je na ľuďoch, ktorý udržujú master branch aby rozhodli, či chcú pull request spojiť s master branchom alebo nie.

Dôvod použitia – Hlavný dôvod, pre použitie nástroja Github, bola jednoduchá kolaborácia medzi zariadeniami, na ktorých prebiehal vývoj a taktiež uložené predošlé verzie kódu v prípade, že sa niečo pokazí.

9.2 Amazon Web Services (AWS)

Podľa dokumentácie [41] Amazon Web Services je služba, ktorá ponúka viac ako 140 služieb cloudových produktov. Funguje to tak, že Amazon Web Services má svoje servery a zákazník si prenajíma určitý výkon servera službou, ktorú potrebuje. Niektoré z ponúkaných služieb sú úložisko, databázy, analýza, vývojárske nástroje atď. Najväčšia výhoda je „pay-as-you-go“ (zákazník platí len za to, čo a koľko používa). Niektoré z výhod používania „cloud computingu“:

- Platíš len za to, čo využívaš
- Lacnejšie ako v prípade budovania a udržovania vlastných serverov v prípade malých firiem
- Jednoduché úprava služby podľa potreby (zväčšenie kapacity disku, zvýšenie výkonu atď.)
- Jednoduché zavedenie aplikácií

Z celého balíku služieb ponúkaných na AWS bola v práci vybraná nasledujúca.

9.2.1 DynamoDB

Rýchla a flexibilná NoSQL databáza. NoSQL databáza ukladá údaje v iných formátoch ako tabuľka (SQL databázy). Ponúka všetky služby aké by mali databázy poskytovať : CRUD (vytvor, čítaj, ulož, zmaž), možnosť zálohy, dotazy atď.

9.2.1.1 Zabezpečenie

Zabezpečenie dát v pokoji

Zabezpečenie dát v pokoji je použitím 256-bit Advanced Encryption Standard (AES-256). Pri vytváraní tabuľky alebo pri prepínaní šifrovania má užívateľ na výber z troch typov tzv. „customer master keys“: kľúč vlastnený AWS (kľúč vlastní DynamoDB), kľúč spravujúci AWS (kľúč uložený v účte zákazníka a spravovaný službou KMS) a kľúč spravujúci užívateľom (kľúč je uložený a vytvorený, vlastnený a spravovateľný užívateľom). [42]

Zabezpečenie dát pri prenášaní po sieti

Zabezpečenie pri prenášaní je pomocou HTTPS protokolu, ktorý chráni sieťový prenos pomocou šifrovania Secure Sockets Layer (SSL) / Transport Layer Security (TLS). [43]

Dôvod použitia – Skúmanie a hľadanie databázy, ktorá by vyhovovala požiadavkám trvalo niekoľko hodín. Požiadavky boli 2: čo najlacnejšie (najlepšie zadarmo) a aby mala jednoduchú implementáciu. Týmto požiadavkám najviac vyhovovala databáza DynamoDB na AWS. Obe požiadavky boli splnené lepšie ako bolo zadané. Pri tak malom množstve dotazov na tabuľku sa momentálne neplatí nič a v C# má DynamoDB svoju knižnicu, čiže ani implementácia nebola taká zložitá.

9.3 Firebase – Google

Google Firebase je backend nástroj pre podporu mobilných aplikácií a web stránok pomocou rôznych backend služieb.

Cenník Firebase služieb je na báze „pay as you go“ a ceny sú škálovateľné na základe toho, aké služby a v akých množstvách (dotazy, GB atď.) sú potrebné. Väčšina služieb je aj do nejakej miery poskytovaná zadarmo. [44]

Podľa ponuky [45] sa Firebase služby delia do troch hlavných skupín: vývoj, vydanie a monitorovanie a „engage“.

9.3.1 Vývoj

Podľa ponuky [46] služieb pre vývoj sú k dispozícii nasledujúce:

Cloud Firestore – služba slúžiaca na skladovanie dát v cloude, synchronizovanie dát medzi online a offline zariadeniami a získavanie dát rýchlymi dotazmi

Realtime Database - služba slúžiaca na ukladanie a synchronizovanie dát v JSON formáte medzi užívateľmi v takmer reálnom čase alebo offline so silným užívateľským zabezpečením

Remote Config – dynamická správa a optimalizácia užívateľskej skúsenosti v produkcií

Firebase ML – výkonné strojové učenie s API pripraveným k použitiu ale aj podporou pre nasadenie vlastného modelu

Cloud Functions – služba na serverovú logiku aplikácie bez potreby nastavovania vlastného servera

Authentification – služba pre poskytnutie jednoduchého prihlasovanie užívateľa s pomocou služieb tretích strán

Hosting – rýchle a bezpečné hostovanie web stránok

Cloud Storage – služba na ukladanie užívateľského obsahu

9.3.2 Vydanie a monitorovanie

Podľa ponuky [47] služieb pre vydanie a monitorovanie sú k dispozícií nasledujúce:

Crashlytics – služba na zaznamenávanie a opravu stability problémov s aplikáciou v reálnom čase

Performance Monitoring – prehľad k výkonu aplikácie a k rýchlemu riešeniu problémov

Test Lab – zachytenie problémov pred vydaním a validovanie aplikácie na fyzických aj virtuálnych zariadeniach, ktoré simulujú prostredie

App Distibution – jednoduchá distribúcia aplikácie pre testerov na dosiahnutie potrebnej odozvy

9.3.3 Engage

Podľa ponuky [48] služieb pre „engage“ sú k dispozícií nasledujúce:

Predictions – strojové učenie na predikovanie užívateľskej interakcie

A/B Testing – experimenty na otestovanie nápadov a vplyv na kľúčové metriky

Cloud Messaging – infraštruktúra na poskytnutie spoľahlivého odosielania a prijímanie push notifikácií

Dynamic Links – prepojenie užívateľov na nájdenie a zdieľanie obsahu

In-App Messaging – povzbudenie užívateľov na splnenie akcie v aplikácií pomocou cielených správ

Dôvod použitia – Zo všetkých spomínaných služieb Firebase boli vybraté služby Cloud Messaging a Remote Config. Služba Cloud Messaging v aplikácií je využívaná na prijímanie a odosielanie push notifikácií. Služba Remote Config je využívaná na kontrolovanie, či má užívateľ aktuálnu verziu aplikácie. Ak nemá, tak pri spustení dostane upozornenie, že je k dispozícií aktualizácia.

9.4 Google Play Console

Google Play Console je služba, ktorej hlavná funkcia je dostať aplikáciu na obchod Google Play. Pri prvotnom zavádzaní aplikácie je potrebné dodať rôzne informácie, vyplniť rôzne typy dotazníkov aby Google vedel aplikáciu sprístupniť pre dané publikum ľudí. Ide hlavne o reštrikcie vzhľadom na vek. Taktiež je treba nahrať .apk ale .aab súbor aplikácie, ktorú chceme nahrať. Jedna z najpodstatnejších vecí pri nahrávaní aplikácie je vybratie kanála, do akého chceme aplikáciu nahrať. Jestvuje možnosť na výber z týchto: ostrá verzia, otvorené testovanie, uzatvorené testovanie, interné testovanie a predbežná registrácia. Po zavedení aplikácie má vývojár k dispozícií množstvo informácií. Informácie o tom, na akých konkrétnych zariadeniach je aplikácia podporovaná, ale aj rôzne štatistiky ohľadom používaní aplikácie užívateľmi.

Dôvod použitia – Jednoduché zavádzanie aplikácie medzi testerov.

9.5 Payme

Payme je služba, ktorá slúži na vytváranie odkazov, ktoré slúžia na zjednodušenie vytvárania platobného príkazu. Jeden užívateľ vygeneruje odkaz zadaním potrebných údajov na stránke alebo v aplikácií. Výstupom je odkaz, ktorý druhému užívateľovi odošle nejakým komunikačným kanálom. Pri kliknutí na tento odkaz v mobile je užívateľ presmerovaný do mobilného internet bankingu a podľa údajov zadaných užívateľom, ktorý tento odkaz vytvoril sa mu automaticky vypíšu informácie vo formulári na zaplatenie platobného príkazu. V prípade, že užívateľ nemá banku alebo nemá nainštalovanú aplikáciu mobilného internet bankingu, ktorá toto presmerovanie podporuje, tak je mu otvorená stránka v prehliadači s QR kódom, po ktorom naskenovaní dané informácie potrebné na vytvorenie platobného príkazu získa. [49]

Dôvod použítia – Jednoduchý spôsob zrealizovania platobného príkazu

9.6 Don’t kill my app – API

Toto API poskytuje informácie o tom, aké nastavenia povoliť alebo zmeniť aby mohla fungovať na pozadí. Vstupný parameter do tohoto API je jeden a to konkrétne výrobca smartphónu. [50]

Dôvod použítia – Zobrazenie nastavení, ktoré je potrebné zmeniť alebo povoliť v operačnom systéme na fungovanie opakujúcich sa notifikácií (12.5).

10 NÁVRHOVÉ VZORY, TECHNIKY A ARCHITEKTÚRY

10.1 Model-View-ViewModel Pattern

Podľa dokumentácie [51] Model-View-ViewModel vzor (ďalej aj MVVM) pomáha oddeliť biznis a prezentačnú logiku aplikácie od užívateľského rozhrania. Takéto oddelenie aplikačnej logiky od užívateľského rozhrania pomáha pri testovaní, rozširovaní a udržiavaní aplikácie. Taktiež použitie MVVM vzoru pomáha znovu použiteľnosti kódu a zjednodušuje kolaboráciu medzi vývojármi a návrhármi užívateľských rozhraní.

Pri používaní MVVM vzoru rozdeľuje grafické rozhrania a prezentačnú a biznis logiku do troch oddelených tried: view, view model a model.

Nasledujúce podkapitoly sú tiež čerpané z dokumentácie [51].

10.1.1 View trieda

Úlohou View triedy je:

- Zobraziť grafické elementy ako stránky, elementy (Label, Buttoin atď), alebo DataTemplate.
- View sa odkazuje na ViewModel pomocou špeciálne určenej vlastnosti (property). Vlastnosti a príkazy užívateľa sú obstarávane pomocou ViewModelu.
- View môže upravovať zobrazovanie dát z viewmodelu ako napríklad použitie konvertera na naformátovanie dát na zobrazenie alebo použitie validácie na vstup užívateľa.
- View definuje správanie grafického rozhrania ako sú animácie, prechody atď.
- Kód na pozadí pre View triedu môže implementovať správania, ktoré je tažké popísať v XAML.

10.1.2 View Model trieda

Úlohou View Model triedy je:

- View Model typicky nie je priamo napojený na konkrétny View. View Model implementuje vlastnosti (property) a príkazy (commandy), na ktoré sa View viaže.
- View Model oznamuje zmeny pomocou udalosti INotifyPropertyChanged alebo INotifyCollectionChanged.

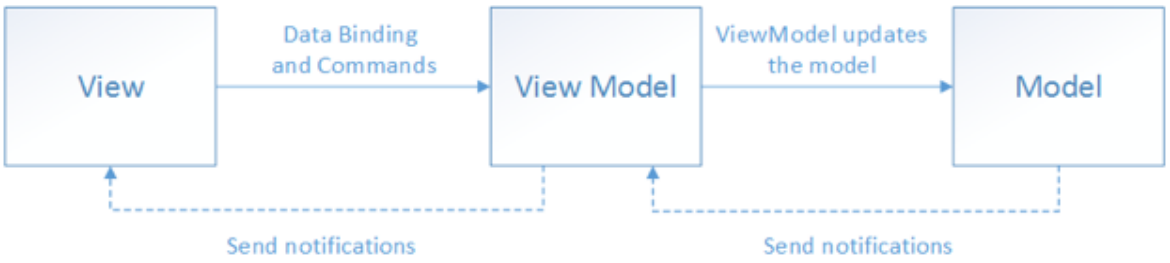
- View Model koordinuje interakciu View s Modelom. Môže manipulovať dáta pre jednoduchšie spracovanie vo View a môže implementovať dodatočné vlastnosti (property), ktoré sa nenachádzajú v Modeli.
- View Model môže definovať logické stavy, ktoré View prezentuje užívateľovi.

10.1.3 Model trieda

Úlohou Model triedy je:

- Model triedy enkapsulujú aplikačné dáta a biznis logiku.
- Model je zodpovedný za správu a konzistentnosť dát.
- Model triedy sa neodkazujú na View priamo a nemajú žiadnu závislosť na tom, ako sú implementované.

Ukážka prepojenia medzi jednotlivými triedami je na obrázku 14.



Obrázek 14 MVVM vzor [52]

10.2 Model-View-Controller pattern

Podľa článku [53] Model-View-Controller (MVC) je vzor, ktorý rozdeľuje aplikáciu do troch logických častí – Model, View a Controller. Každá časť zodpovedá špecifickému vývojovému aspektu aplikácie. MVC oddeľuje biznis logiku od prezentačnej vrstvy. V súčasnosti je tento vzor používaný hlavne pri webových stránkach a mobilných aplikáciách.

10.2.1 View

View je časť aplikácie, ktorá reprezentuje prezentáciu dát. View je vytvárané z dát poskytnutých Model-om. View obsahuje všetky grafické komponenty viditeľné koncovým užívateľom. [53]

10.2.2 Controller

Controller má za úlohu zabezpečovať interakciu užívateľa. Controller informuje Model a View na potrebné zmeny. Controller odosieľa príkazy Model-u na zmenenie stavu. [53]

10.2.3 Model

Model obsahuje uložené dáta a logiku s nimi spojenú. Reprezentuje dáta, ktoré sú posielané medzi komponentami Controllera. [53]

10.3 Dependency Injection

Podľa článku [54] Dependency Injection je vzor, ktorý umožňuje vytvárať závislé objekty mimo triedu a následne poskytnúť tieto objekty triedy pomocou troch variantov. Použitím Dependency Injection presúvame vytvorenie a nalinkovanie objektu na danú triedu. Dependency Injection vzor zahŕňa tri typy tried:

Client trieda (závislá trieda) - je trieda závislá na service triedu

Service trieda (závislosť) – je trieda, ktorá poskytuje služby client triede

Injector trieda – je trieda, ktorá „injektuje“ (vkladá) objekt service triedy do client triedy

Typy Dependency Injection sa líšia v spôsobe „injektovania“ (vloženia) objektu. Tie sú nasledovné:

Constuctor injection – „injektovanie“ (vloženie) prebieha prostredníctvom konštruktoru client triedy

Property injection – „injektovanie“ (vloženie) prebieha prostredníctvom verejnej property v client triede

Method injection – pri tomto type musí client trieda implementovať rozhranie, ktoré obsahuje metódu pomocou ktorej bude objekt „injektovaný“ (vložený)

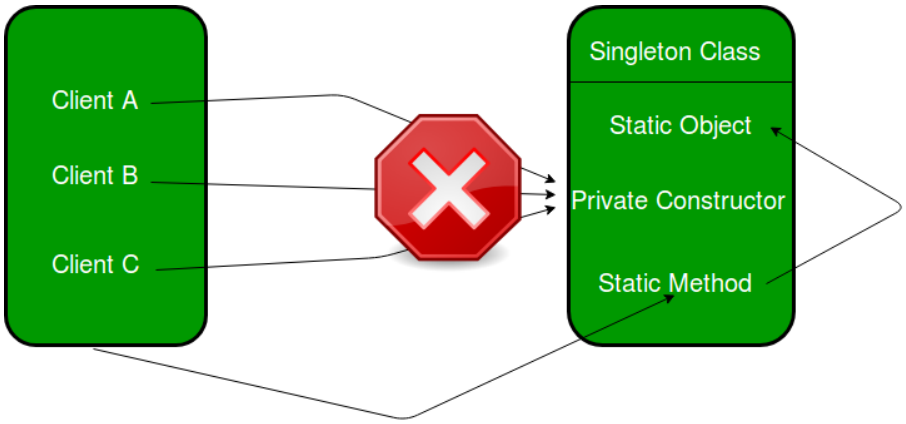
10.4 IoC Container

IoC Container implementuje automatický dependency injection. Spravuje vytvorenie objektu a jeho životnosť ako aj „injektovanie“ (vloženie) závislostí do tried. IoC Container vytvára objekty špecifických tried a „injektuje“ (vkladá) všetky závislosti pomocou konštruktoru, property alebo metódy v reálnom čase a likviduje ich vo vhodný čas. Tým pádom odpadáva vývojárom manuálne vytváranie a spravovanie objektov. [55]

10.5 Singleton design pattern

Singleton je vzor, ktorý znemožňuje triede mať viacero instancií. [56]

Ukážka prístupu k Singleton classe je na obrázku 15.



Obrázek 15 Singleton design pattern [56]

Singleton môže byť inicializovaný dvoma spôsobmi. Tieto spôsoby sú:

Včasná inicializácia

Pri tomto type je trieda so singletonom inicializovaná aj keď nie je používaná. Je inicializovaná, keď sa načítava trieda. [56]

Lenivá inicializácia

Pri tomto type je trieda so singletonom incializovaná len vtedy, keď je potrebná. [56]

Existuje niekoľko metód implementácie v jazyku C#. Ja som vybral iba niektoré.

„simple thread-safety“

```
public sealed class Singleton
{
    private static Singleton instance = null;
    private static readonly object padlock = new object();

    Singleton()
    {
    }

    public static Singleton Instance
    {
        get
        {
            lock (padlock)
            {
                if (instance == null)
                {
                    instance = new Singleton();
                }
                return instance;
            }
        }
    }
}
```

Obrázek 16 „simple thread-safety“ Singleton [57]

Táto implementácia je bezpečná z pohľadu viac-vláknového programovania. Vlákno zamkne zdieľaný objekt a potom zisťuje, či už instancia vytvorená bola. V prípade, že nebola, tak ju vytvorí. [57]

„fully lazy instantiation“

```
public sealed class Singleton
    {
        private Singleton()
        {
        }

        public static Singleton Instance { get { return Nested.instance; } }

        private class Nested
        {
            // Explicit static constructor to tell C# compiler
            // not to mark type as beforefieldinit
            static Nested()
            {
            }

            internal static readonly Singleton instance = new Singleton();
        }
    }
```

Obrázek 17 „fully lazy instantiation“ Singleton [57]

Pri tomto type je vytvorenie instancie spustené pri prvej referencií statickej „instance“ v „static Nested“. Toto je vyvolané jedine z „Instance“. [57]

„.NET 4's Lazy<T> type“

```
public sealed class Singleton
    {
        private static readonly Lazy<Singleton>
            lazy =
            new Lazy<Singleton>
                (() => new Singleton());

        public static Singleton Instance { get { return lazy.Value; } }

        private Singleton()
        {
        }
    }
```

Obrázek 18 „.NET 4's Lazy<T> type“ Singleton [57]

Pri tejto metóde sa využíva „System.Lazy<T>“. To, čo je treba spraviť, je zavolanie konštruktora s delegátom – to je jednoducho dosiahnuteľné s pomocou lambda výrazov. [57]

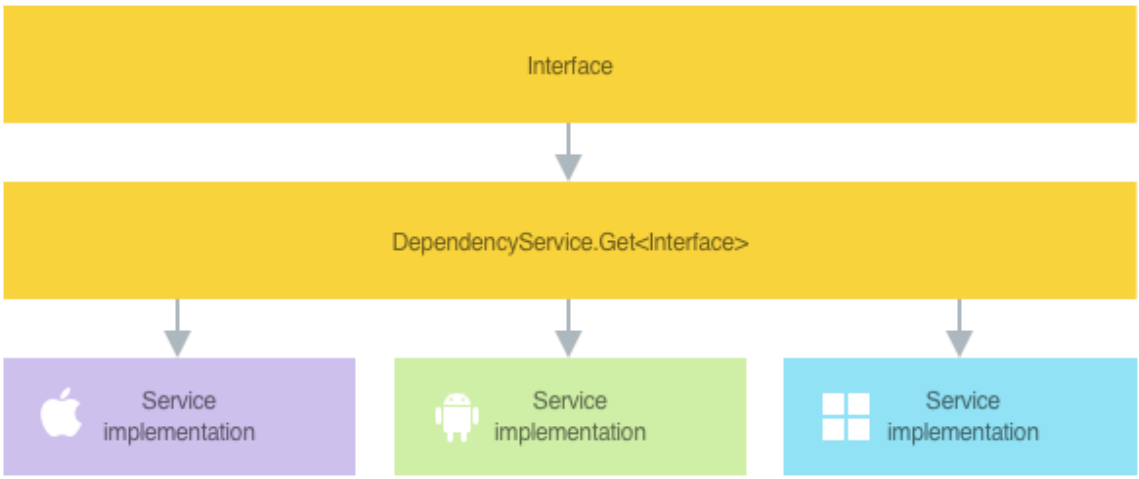
10.6 Dependency Service

Podľa dokumentácie [58], Dependency Service trieda je služba, ktorá dokáže vyvolať natívny kód špecifických platforiem zo zdieľaného zdrojového kódu.

Proces použítia služby Dependency Service je:

- Vytvoriť rozhranie pre naše potreby
- Implementovať tento vytvorený interface na potrebných platformách
- Zaregistrovať triedu na použitie Dependency Service
- Vyvolať Dependency Service zo zdieľaného zdrojového kódu

Proces funkcionality v diagrame je na obrázku 19.



Obrázek 19 Dependency Service – funkčnosť [58]

II. PRAKTICKÁ ČÁST

11 NÁVRH APLIKÁCIE

V tejto kapitole sa nachádzajú funkčné a nefunkčné požiadavky na základe ktorých bola aplikácia vyvíjaná.

11.1 Funkčné požiadavky

V tejto časti sa nachádza zoznam funkčných požiadaviek. Funkčné požiadavky sú tie, ktoré definujú funkcie, ktoré má aplikácia obsahovať

Zoznam funkčných požiadaviek:

1. **Výpis všetkých dlhov užívateľa**
2. **Prijímať push notifikácie**
3. **Prihlásenie užívateľa**
4. **Registrácia užívateľa**
5. **Obnovenie hesla**
6. **Zmena osobných údajov (email, IBAN)**
7. **Vkladať dlhy –** vložené dlhy užívateľom budú posúdené administrátorom
8. **Opakované notifikácie –** opakujúca sa notifikácia (v prípade splnených požiadaviek pre typ zariadenia) keď dlhujúci užívateľ dlh odoslal
9. **Zobrazenie požiadaviek na opakované notifikácie –** presný postup čo a ako v systéme na ktorom je aplikácia nainštalovaná povoliť pre funkčnosť
10. **Prepínanie medzi zobrazením dlhov**
11. **Prepínanie medzi „User“ a „Admin“ módom –** v prípade, že užívateľ je v administrátor v nejakej skupine
12. **Zobrazovanie dlhov celej skupiny administrátorovi danej skupiny**
13. **Mazanie dlhov všetkých užívateľov administrátorom danej skupiny**
14. **Prijímanie a zamietanie dlhov adminom, ktoré pridajú užívatelia**
15. **Rozlišovať medzi odoslanými a neodoslanými**
16. **Témy –** Užívateľ si môže vybrať medzi svetlou a tmavou verziou aplikácie
17. **Filtrovanie dlhov –** filtrovanie dlhov podľa parametrov ako dlhujúci, minimálna a maximálna suma
18. **Vyhľadávanie medzi dlhmi**
19. **Odoslanie dlhov –** možnosť odoslať informáciu o zaplatenom dlhu
20. **Vymazanie dlhov –** možnosť vymazať už prijaté dlhy

21. **Presmerovanie na zaplatenie dlhu –** v prípade splnených požiadavkou služby pay.me
22. **Zobrazenie podmienok používania a zásady ochrany osobných údajov**
23. **Vytvorenie a spravovanie skupiny**
24. **Pripojenie do skupiny**

11.2 Nefunkčné požiadavky

V tejto časti sa nachádza zoznam nefunkčných požiadaviek. Nefunkčné požiadavky sú tie, ktoré zabezpečujú použiteľnosť a efektivitu.

Zoznam nefunkčných požiadaviek:

1. **Multiplatformová aplikácia**
2. **GDPR –** Užívateľ by mal mať informácie o údajoch, ktoré o ňom prevádzkovateľ uchováva. Užívateľ môže svoj účet zmazať spolu s údajmi, ktoré sú s ním asociované.
3. **Zabezpečenie dát užívateľa**
4. **Cena za správu aplikácie by mala byť čo najmenšia**

12 KĽÚČOVÉ PRVKY APLIKÁCIE

12.1 Práca s databázou

12.1.1 Pripojenie k databáze

Na komunikáciu s databázou DynamoDB sú využívané služby knižnice **AWSSDK.DynamoDBv2**. Táto knižnica ponúka viacero možností ako pristupovať k databáze. V aplikácií sa pracuje s možnosťou pristupovania pomocou .NET: Object Persistence Model. Podľa dokumentácie [59], tento prístup nám umožňuje namapovať triedy na strane klienta s DynamoDB databázou. Vstupným bodom do databáze je trieda *DynamoDBContext,* pomocou ktorej voláme dotazy CRUD. Táto trieda na vytvorenie instancia potrebuje objekt triedy *BasicAWSCredentials* a región, v ktorej sa databáza nachádza. Informácie potrebné na vytvorenie objektu *BasicAWSCredentials* (AWS Access Key a AWS Secret Key) uložené v separátnom súbore kvôli zabezpečeniu.

12.1.2 Deklarovanie tabuľky pomocou .NET: Object Persistence Model

Ako je možné vidieť na obrázku 20, atribút DynamoDBTable označuje tabuľku a parameter tohto atribútu je požadovaný názov tabuľky. Atribút DynamoDBHashKey označuje primárny kľuč tabuľky a DynamoDBIgnore je atribút, ktorý sa v tabuľke nachádzať nebude, používa sa iba lokálne.

```
[DynamoDBTable("DLHY")]
    public class DLHY
    {
       [DynamoDBHashKey]
        public string ID { get; set; }
        public string KTO { get; set; }
        public string KOMU { get; set; }
        public double KOĽKO { get; set; }
        public string KEDY { get; set; }
        public string Popis { get; set; }
        public string Skupina { get; set; }
        public bool Potvrdene { get; set; }
        public bool Odoslany { get; set; }
        [DynamoDBIgnore]
        public bool Dlhujúci { get; set; }
    }
```

Obrázek 20 Model Užívateľa

12.1.3 Ukážka pridania nového dlhu

Ako je možné vidieť na obrázku 21, pridávanie nového dlhu je veľmi jednoduché. V podstate je vytvorený objekt *novyDlh* kde do konštruktora sú priradené všetky potrebné atribúty, ktoré má dlh obsahovať. Po tom, ako objekt *novyDlh* je vytvorený, je do databáze uložený pomocou metódy *SaveAsync* z *context*.

```
public async Task PridajDLH(string kto, string komu, double koľko, DateTime kedy,
string popis, string skupina, bool potvrdene)
        {
            using (DynamoDBContext context = new DynamoDBContext(client))
            {
                Guid g = Guid.NewGuid();

                DLHY novyDlh = new DLHY
                {
                    ID = g.ToString(),
                    KTO = kto,
                    KOMU = komu,
                    KOĽKO = koľko,
                    KEDY = kedy.ToShortDateString(),
                    Skupina = skupina,
                    Popis = popis,
                    Potvrdene = potvrdene
                };

                await context.SaveAsync(novyDlh);
            }
        }
```

Obrázek 21 Pridanie nového dlhu

12.1.4 Ukážka vrátenia dlhov pre konkrétnu kombináciu užívateľov

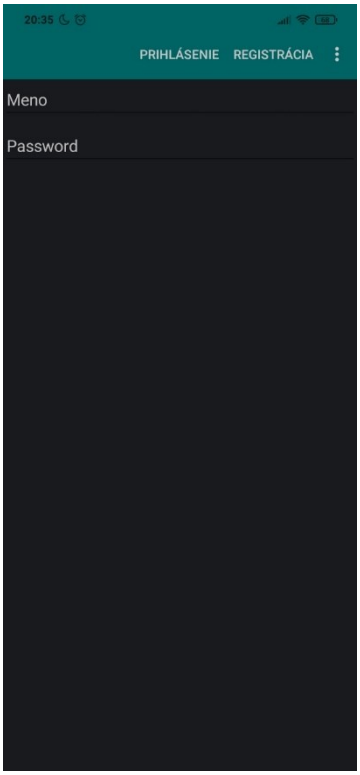
V DynamoDB nám k vytváraniu dotazov slúžia dve metódy: Query a Scan. Rozdiel medzi nimi je ten, že Query prehľadáva pomocou primárnej alebo sekundárnej partície, zatiaľ čo Scan prehľadáva celú tabuľku [60]. V celej aplikácií je používaná metóda Scan na vytváranie dotazov. V tejto ukážke sú hľadáne všetky dlhy pre konkrétnu kombináciu užívateľov. Do objektu config sú pridané *ScanCondition* kde prvý parameter je názov property z modelu, druhý parameter je operátor a tretí parameter je hodnota. V prípade z obrázku 22, sú *ScanCondition* použité 2. V prvej kontolujeme, či sa property *KTO* zhoduje s fieldom *meno_dlznika*. V druhej je kontrolovaná property *KOMU* či sa zhoduje s fieldom *meno_komu*. Výstupom z databázy je list, v ktorom sa nachádzajú všetky dlhy, ktoré spĺňajú zadané podmienky.

```
public async Task<List<DLHY>> VratDLHY(string meno_dlznika, string meno_komu)
        {
            using (DynamoDBContext context = new DynamoDBContext(client))
            {
                var config = new List<ScanCondition>();
                config.Add(new ScanCondition("KTO", ScanOperator.Equal,
meno_dlznika));
                config.Add(new ScanCondition("KOMU", ScanOperator.Equal, meno_komu));
                var response = await
context.ScanAsync<DLHY>(config).GetRemainingAsync();
                return response;
            }
        }
```

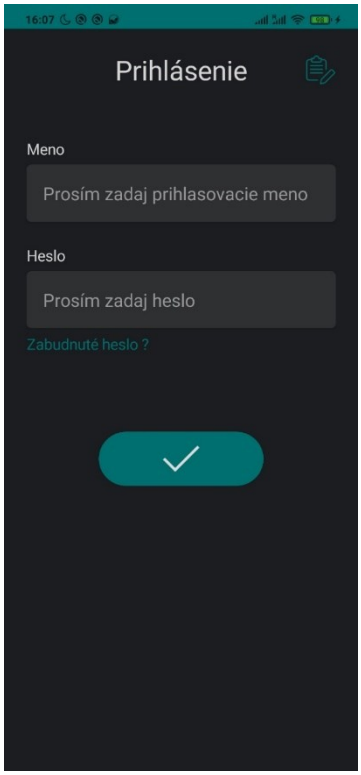
Obrázek 22 Ukážka vrátenia dlhov

12.2 Užívateľské rozhranie

Pri začiatku vývoja tejto aplikácie boli na užívateľské rozhranie použité prvky dostupné v Xamarin.Forms a veľmi na to nebol braný ohľad. Sústredenie bolo hlavne na veci, ktoré sa týkali správnej funkčnosti aplikácie z pohľadu backendu. Pre porovnanie sa nachádza obrázok 23 z predošlej verzie aplikácie. Na to, aby bolo užívateľské rozhranie zmenené z podoby v akej sa nachádzalo do podoby v akej je teraz, bolo potrebné použiť prístup, ktorý sa volá “custom renderers”. Používanie týchto “custom renderers” môže slúžiť k malým zmenám prvkov alebo aj ku kompletnému vytvoreniu nových prvkov. Pri tomto prístupe sa v zdieľanom kóde vytvorí hlavná trieda. Na túto triedu potom necháme vyexportovať zmenený grafický prvok, ktorý je nutné vytvoriť vo všetkých špecifikáciách platforiem, ktoré používame. V aplikácií je tento prístup využitý hlavne na upravenie vzhľadu elementov pre tmavú tému. Tieto elementy sú výber dátumu (date picker), vyberač (picker) a vstup od užívateľa (entry).



Obrázek 23 Prihlásenie (ver. 8)



Obrázek 24 Prihlásenie (ver. 10.1)

12.3 Zabezpečenie aplikácie

12.3.1 Autentifikácia

Existuje mnoho spôsobov, ako riešiť autentifikáciu. Aj keď by bolo možné použiť službu ako Firebase Authentification alebo AWS Cognito, tak bola spravená vlastná implementácia autentifikácie. Užívateľ pri registrácií zadá požadované prihlasovacie meno, heslo a email. Po klient verifikácií hesla a verifikácií zo serverom, že sa užívateľ s takým menom už v databáze nenachádza, heslo sa zahashuje so saltom s pomocou funkcií z namespace *System.Security.Cryptography*. Heslo je zahashované pomcou hashing algoritmu Password-Based Key Derivation Function 2 (PBKDF2), ktorý používa trieda Rfc2898DeriveBytes [61]. Toto zahashovane heslo a salt sú potom uložené v databáze. Pri prihlásení užívateľa je salt a zahashovane heslo z databázy vytiahnuté. Ak sa vložené heslo používateľa, ktoré sa spolu so saltom z databázy zahashuje a zhoduje sa so zahashovaným heslom z databázy, užívateľa aplikácia úspešne prihlási.

12.3.2 Uloženie prihlasovacích údajov v aplikácií

Pre zjednodušenie prihlasovanie sú prihlasovacie užívateľa automaticky zapamätané aplikáciou po prihlásení. Na tieto potreby som použil triedu *SecureStorage* z knižnice Xamarin.Essentials. Táto trieda zabezpečuje bezpečné uloženie údajov v tvare kľúč ako identifikátor pre načítanie a hodnoty, ktorú chceme uložiť. Pri prihlásení užívateľa je údaj o prihlasovacom mene a hashe hesla uložený pomocou tejto triedy.

12.3.3 Zabezpečenie API kľúčov

API kľúče potrebné pre komunikáciu s databázou a s Firebase službami mám uložené v separátnom súbore vo formáte JSON. Tento súbor má nastavenú *Build Action* ako *Embedded resource* čo znamená, že je tento súbor zabudovaný v programe.

```
public static string Load(string whatToLoad)
        {
            var assembly = Assembly.GetExecutingAssembly();

            var resourceName = "DLZOBY.appsettings.json";

            using (Stream stream = assembly.GetManifestResourceStream(resourceName))
            using (StreamReader reader = new StreamReader(stream))
            {
                string json = reader.ReadToEnd();

                JObject array = JObject.Parse(json);

                return array[whatToLoad].ToString();
            }
        }
```

Obrázek 25 Ukážka načítania API kľúča z JSON súboru

Ako je vidieť na obrázku 25 pomocou *assembly.GetManifestResourceStream(resourceName)* získame *stream* z požadovaného súboru. Tento *stream* je následne pomocou *StreamReader* prečítaný a uložený do stringu *json*. Z tohto stringu je následne vytvorený *JObject* pomocou *JObject.Parse(json)*. Následne je požadovaný API kľúč s názvom *whatToLoad* v tomto objekte nájdený a vrátený.

12.3.4 NoSQL Injection

NoSQL Injection je spôsob útoku na databázu, ktorá nepoužíva SQL syntax na vykonávanie dotazov. Tento typ útoku funguje tak, že užívateľ do vstupného poľa zadá špeciálne znaky pomocou ktorých sa snaží získať údaje, ktoré mu nemajú byť prístupné. [62]

V aplikácií sa vstupných polí pre vstup užívateľa nachádza viac, ale väčšina z nich pracuje s údajmi, ktoré sú už lokálne stiahnuté z databázy. Jediné polia, ktoré majú priamy prístup do databáza sú z registrácie predovšetkým užívateľské meno. Spôsob ošetrenia proti tomuto útoku je taký, že užívateľovi nie je povolené mať žiadne špeciálne znaky v mene.

12.4 Push notifikácie

Na posielanie push notifikácií bola vybratá službu Firebase Cloud Messaging. Podľa dokumentácie [63] ponúka táto služba dve typy správ, ktoré sú odosielané v týchto push notifikáciách. „Notifikáčná správa“ sa zobrazuje na koncovom zariadení automaticky v mene aplikácie. Tento typ správy obsahuje niekoľko preddefinovaných kľúčov ako title a body. Druhý typ správ je „dátová správa“. Tento typ správ nie je na koncových zariadeniach zobrazovaný automaticky a taktiež neobsahuje žiadne kľúče s rezervovanými menami. Tieto správy musí spracovať na zobrazenie aplikácia. Informácia, na aké zariadenie má byť konkrétna push notifikácia odoslaná, je definovaná tokenom. Pri registrácií užívateľa je tento token ukladaný do databázy a pomocou neho sú následne notifikácie odosielané. V aplikácií sú využívané „notifikačné správy“ na odosielanie informácií o pridanom dlhu adminovi alebo užívateľom. „Dátové správy“ sú využívané na spustenie opakujúcich sa notifikácií. Viac o tomto sa nachádza v kapitole 12.5.

12.5 Opakujúce sa notifikácie

Dôvodom potreby tohto prvku aplikácie bolo informovať užívateľa o tom, že mu dlhujúci užívateľ poslal dlh. Táto notifikácia je zobrazovaná každých 24 hodín od prijatia push notifikácie, ktorá tieto opakujúce sa notifikácie spustí. Konkrétne je to notifikácia typu „dátová správa“. Viac o tomto type správy je v kapitole 12.4. Problém pri týchto opakujúcich sa notifikáciách bol v tom, že predajcovia smartfónov nedovoľujú na svojich zariadeniach bežať aplikáciu na pozadí. Z tohto dôvodu opakujúce sa notifikácie nefungujú automaticky. Užívateľ ich musí zapnúť podľa stránky, ktorá mu vyskočí pri prvom spustení aplikácie. Každému užívateľovi sa zobrazí stránka podľa výrobcu jeho telefónu. Tieto informácie sú zabezpečované vďaka API Don’t kill my app (9.6).

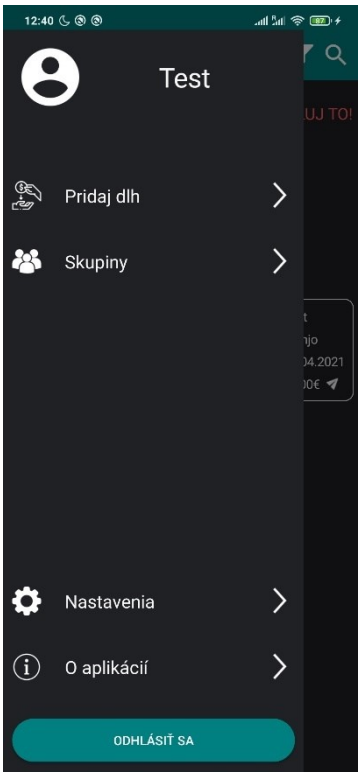
Opakujúce sa notifikácie sa používajú v aplikácií pri jednom prípade použitia. Keď užívateľ odošle dlh tak sa nielen zobrazí informácia o odoslanom dlhu pri konkrétnom dlhu ale v prípade, že užívateľ postupoval podľa návodu na stránke 43 tak sa spustí opakujúca sa notifikácia každých 24 hodín, ktorá mu pripomenie, že mu má prísť dlh.

13 UŽÍVATEĽSKÁ DOKUMENTÁCIA

V tejto kapitole sú popísané jednotlivé okná aplikácie a jednotlivé prvky.

13.1 Navigácia

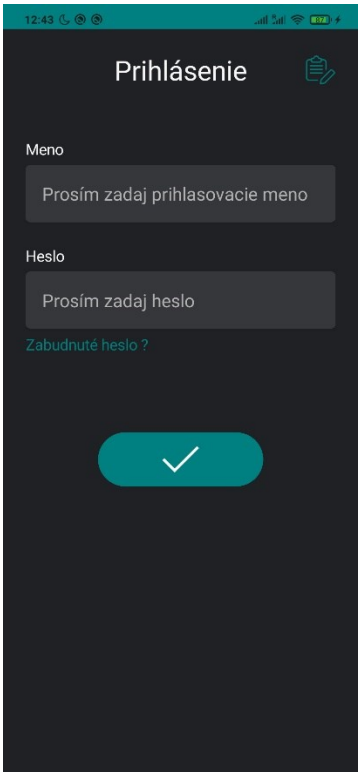
Pri prihlásení užívateľovi je otvorená stránka so zobrazenými dlhmi (13.4) podľa zvolenej možnosti. Všetky ostatné funkcie aplikácie sú schované v navigačnom tzv. hamburger menu. Toto hamburger menu je možné vidieť na obrázku 26.



Obrázek 26 Navigačné menu

13.2 Prihlásenie

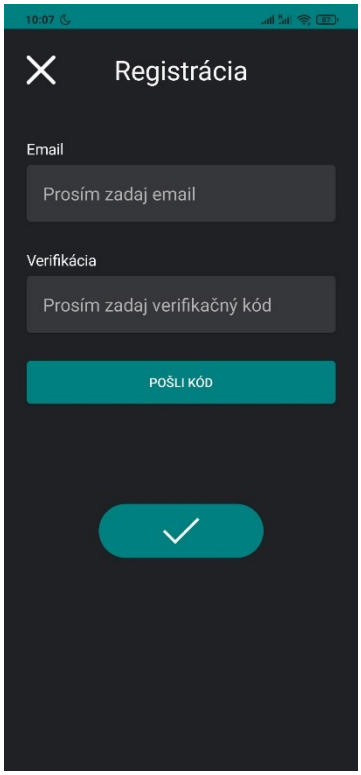
Prihlásenie (obrázok 27) je prvá stránka, ktorá sa užívateľovi zobrazí po nainštalovaní aplikácie. Po zadaní prihlasovacieho mena a hesla je toto heslo zahashované a porovnané s hashom hesla uloženým v databáze. V prípade zhody je užívateľ prihlásený. V prípade nezhody je užívateľ informovaný o tom, že zadané údaje sú zlé.



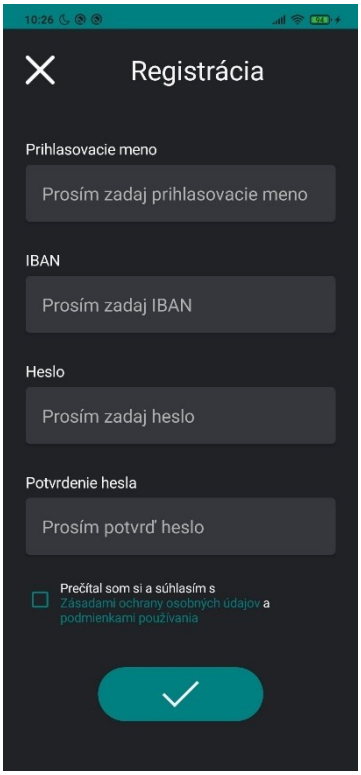
Obrázek 27 Prihlásenie

13.3 Registrácia

Na stránku registrácie sa užívateľ dostane po kliknutí na ikonu v pravom hornom rohu na stránke prihlásenia (13.2). Po kliknutí na túto ikonu sa najskôr zobrazí stránka verifikácie (obrázok 28). Táto stránka slúži na to, aby aplikácia overila, že emailová adresa ktorú chce užívateľ použiť naozaj patrí jemu. Po zadaní emailu musí užívateľ vyžiadať o poslanie kódu pomocou tlačidla „Pošli kód“. Po zadaní kódu a vyhodnotenie aplikácie o správnosti je užívateľovi dovolené pokračovať v registrácií (obrázok 29). Po zadaní všetkých údajov aplikácia overí, či sa zadané heslá zhodujú. V prípade že áno, užívateľ je úspešne zaregistrovaný.



Obrázek 28 Registrácia - verifikácia



Obrázek 29 Registrácia

13.4 Zobrazovanie dlhov

Zobrazovanie dlhov je stránka, ktorá sa užívateľovi zobrazí po úspešnom prihlásení. Je to najdôležitejšia stránka aplikácie, ktorá slúži na zobrazovanie všetkých dlhov užívateľa ako aj na správu týchto dlhov. V aplikácií sa nachádzajú dva módy zobrazovania dlhov – zoznam (obrázok 30) a dlaždice (obrázok 32). Pri móde zoznam sú spočiatku zobrazené najdôležitejšie informácie o dlhoch voči konkrétnemu užívateľovi, a to výsledná čiastka z dlhov. Ako je vidieť, možnosti správy dlhu sú spočiatku skryté. Ako je vidieť na obrázku 31, odokryť ich je možné potiahnutím daného dlhu doprava. Taktiež čiastkové dlhy, z ktorých je výsledná suma dlhu vyrátaná sa zobrazia až po kliknutí na konkrétny dlh. Keď užívateľ je v konkrétnom dlhu ako dlhujúci, tak po potiahnutí doprava má možnosť skopírovať IBAN toho, komu dlhuje. Na rozdiel od zoznamu, dlaždice (obrázok 32) obsahujú všetky informácie bez nutnosti rozkliknutia. Výsledná suma dlhu je vypísaná nad dlaždicami a samotné dlaždice sú čiastkové dlhy. Pre prehľadnejšie zobrazenie dlhov v prípade väčšieho počtu môže užívateľ dlhy filtrovať podľa dostupných filtrov (viz. obrázok 33). Pre prípad, že by aj po vyfiltrovaní bolo dlhov stále veľa, užívateľ má ešte možnosť vyhľadať dlhy podľa nejakého parametru.

13.4.1 Správa dlhov

Užívateľ má možnosť dlhy spravovať. Teraz popíšem jednotlivé funkcie a vysvetlenie.

Správa dlhov dlhujúceho

Odoslanie dlhu - Odoslanie dlhu spôsobí dve veci. Označí dlh ako odoslaný (viz. obrázok 30, tretí dlh) a v prípade, že má príjemca dlhu povolené opakujúce sa notifikácie tak sa mu zapnú a budú mu chodiť každých 24 hodín.

Transakcia - Vytvorí odkaz na zaplatenie pomocou služby payme (9.5). V prípade, že má užívateľ nainštalovanú aplikáciu banky, ktorá túto službu podporuje tak to môže otvoriť priamo v nej a informácie potrebné k zaplateniu dlhu sa mu automaticky vypíšu.

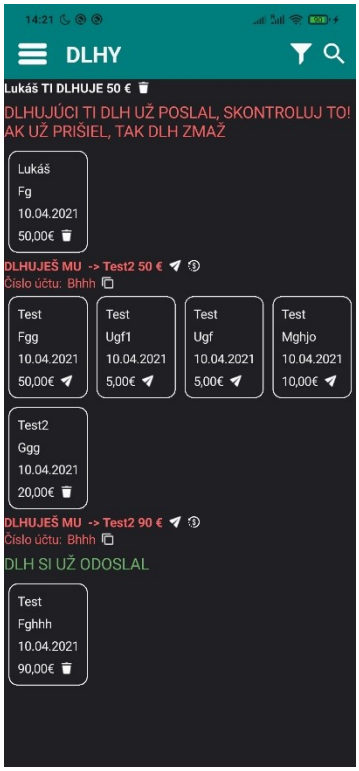
Kopíruj IBAN - Skopíruje do schránky IBAN príjemcu konkrétneho dlhu.

Správa dlhov príjemcu

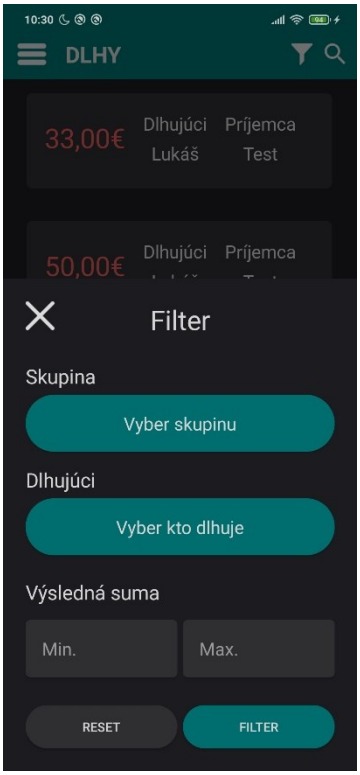
Vymaž - V prípade kliknutia tejto ikony pri konkrétnom čiastkovom dlhu, vymaže tento dlh pokiaľ platil prihlásený užívateľ. Pri kliknutí ikony, ktorá sa nachádza pri celkovej sume sa vymažú všetky čiastkové dlhy.

Obrázek 30 Zobrazovanie dlhov - zoznam

Obrázek 31 Zobrazovanie dlhov – zoznam – správa dlhov



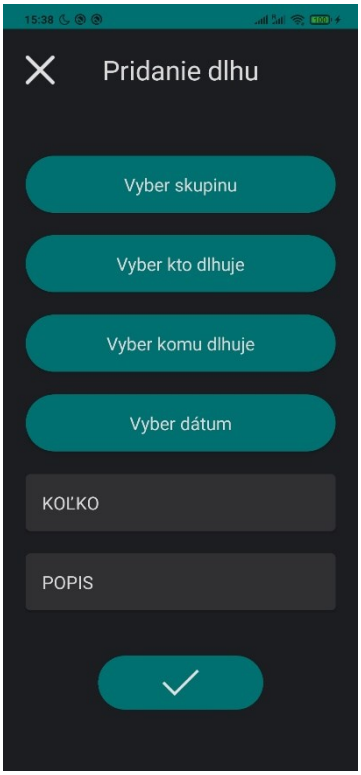
Obrázek 32 Zobrazovanie dlhov - dlaždice



Obrázek 33 Filter

13.5 Pridanie dlhu

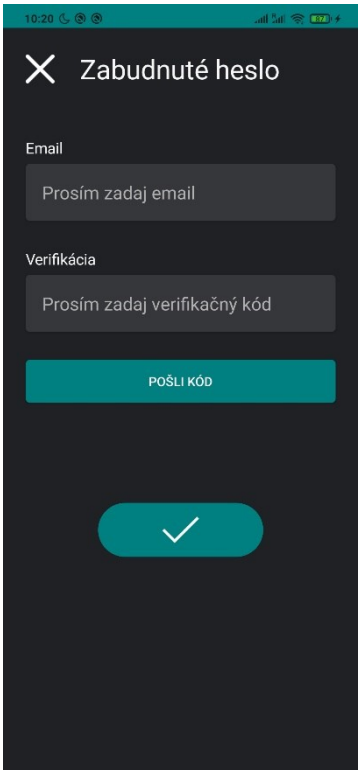
Na stránku pridania dlhu sa užívateľ dostane po kliknutí na „Pridaj dlh“ z navigačného menu (obrázok 26). Po vyplnení všetkých potrebných údajov (viz. obrázok 34) záleží od role užívateľa v skupine, do ktorej dlh vkladal. Ak je v danej skupine užívateľ administrátor tak je dlh vložený automaticky do databáze a informácia o vloženom dlhu je odoslaná dlhujúcemu užívateľovi. V prípade, že užívateľ administrátor nie je, tak dlh je vložený do databázy, ale ešte nie je viditeľný dlhujúcemu užívateľovi. Informácia o pridanom dlhu je odoslaná administrátorovi danej skupiny, ktorý musí daný dlh preveriť. V prípade, že dlh prešiel preverením administrátora, tak je zviditeľnený dlhujúcemu užívateľovi a je o tom informovaný push notifikáciou.



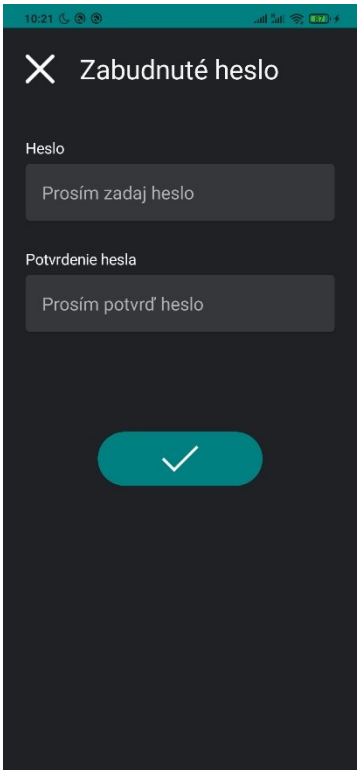
Obrázek 34 Pridanie dlhu

13.6 Zabudnuté heslo

Na stránku zabudnutého hesla sa užívateľ dostane po kliknutí na odkaz „Zabudnuté heslo“, ktorý sa nachádza pod vstupom na heslo na stránke prihlásenia (13.2). Po kliknutí na tento odkaz sa najskôr zobrazí stránka verifikácie (obrázok 35). Táto stránka slúži na to, že účet na ktorom chce užívateľ zmeniť heslo patrí naozaj jemu. Po zadaní emailu musí užívateľ vyžiadať o zaslanie kódu pomocou tlačidla „Pošli kód“. Po zadaní kódu a vyhodnotenie aplikácie o správnosti je užívateľ presmerovaný na stránku so zmenením hesla. Po zadaní hesla a aj potvrdenia hesla prebehne overenie. V prípade, že sa hesla zhodujú, užívateľ úspešne zmenil svoje heslo.



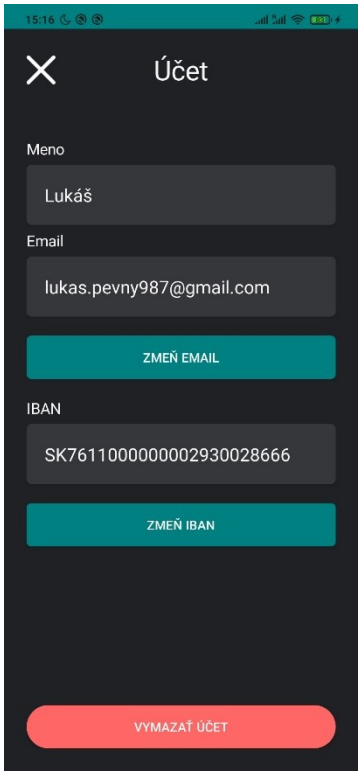
Obrázek 35 Zabudnuté heslo - verifikácia



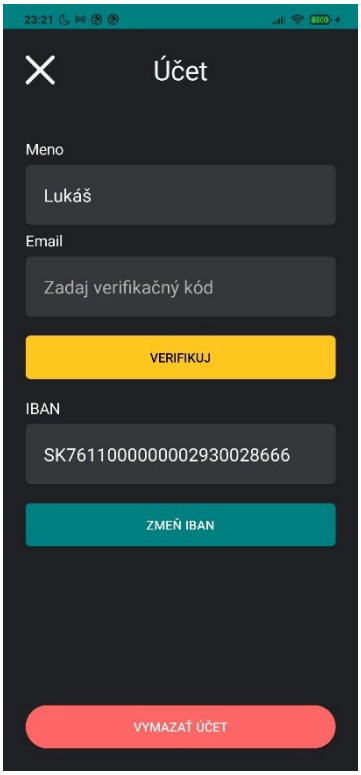
Obrázek 36 Zabudnuté heslo – zmena hesla

13.7 Účet

Na stránku účtu sa užívateľ dostane po kliknutí na ikonu účtu alebo názov účtu z navigačného menu (obrázok 26). Na stránke účtu (obrázok 37) si užívateľ môže zmeniť email a IBAN. Predtým ako užívateľ zmení svoj email, musí prejsť dvomi verifikáciami. Po kliknutí na „ZMEŇ EMAIL“ je na aktuálny email odoslaný verifikačný kód. Toto slúži na overenie, že užívateľovi nebol účet odcudzený. Zároveň sa dynamicky zmení stránka ako je vidieť na obrázku 38. Po zadaní verifikačného kódu a kliknutí na „VERIFIKUJ“ je užívateľ presmerovaný na stránku zmeny emailu ako, ktorá je identická ako stránky obrázku 28 a obrázku 35. Po odoslaní verifikačného kódu na nový email užívateľ daný verifikačný kód zadá. Táto verifikácia jestvuje z takého dôvodu, že užívateľ má prístup k emailu, ktorý chce nastaviť ako nový email. Po úspešnej verifikácií zadaného kódu je užívateľovi email zmenený.



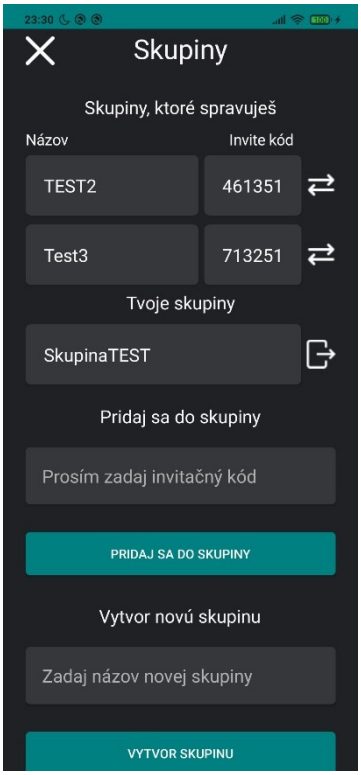
Obrázek 37 Účet



Obrázek 38 Účet – zmena emailu - verifikácia

13.8 Skupiny

Na to, aby užívateľ mohlo nejaké dlhy mať, musí sa v nejakej skupine nachádzať. Táto stránka (obrázku 39) poskytuje niekoľko funkcií ohľadom skupín. „Skupiny, ktoré spravuješ“ je zoznam skupín, v ktorých užívateľ zastáva rolu admina resp. správcu danej skupiny. Ikona pri týchto skupinách znamená presunutie role admina na inú osobu z danej skupiny. Ďalej sa tu nachádzajú „Tvoje skupiny“, kde sa nachádza zoznam všetkých skupín, v ktorých sa užívateľ nachádza (okrem tých, v ktorých si Admin). Ikona pri týchto skupinách znamená opustenie danej skupiny. Skupinu môže užívateľ opustiť jedine vtedy, ak už v danej skupine nemá žiadne dlhy. „Pridaj sa do skupiny“ je funkcia, ktorá užívateľa po vložení „invite code“ do danej skupiny pridá. Funkciou „Vytvor novú skupinu“ ako názov napovedá užívateľ vytvorí novú skupinu v ktorej bude zastávať rolu „admina“.

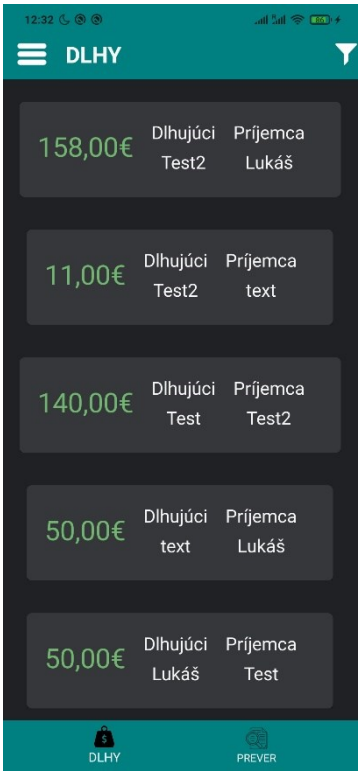


Obrázek 39 Skupiny

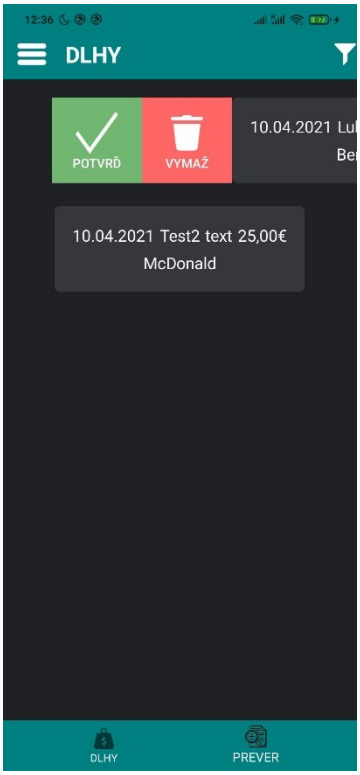
13.9 Admin mód

Do admin módu sa dostane užívateľ v prípade, že je admin nejakej skupiny po kliknutí na „ADMIN MÓD“ v navigačnom menu (obrázok 26). Následne sa užívateľovi otvorí stránka, ktorá má dve okná – „DLHY“ a „PREVER“. Okno „DLHY“ (obrázok 40) obsahuje výpis dlhov všetkých užívateľov skupín, v ktorých je prihlásený užívateľ admin. Admin má možnosť tieto dlhy vymazať. Spôsob zobrazenia týchto dlhov bol spočiatku v aplikácií ako tabuľka. Toto riešenie bolo neprehľadné a mätúce. Terajšie riešenie malo spočiatku slúžiť iba pre admina, ale po premyslení bolo pôvodné zobrazenie dlhov pre jednu osobu zmenené na toto riešenie a pridanie filtra k tomuto zobrazeniu (viz. obrázok 33). V spomínanom filtri sa mimo ostatných možných filtrov nachádza aj vybratie konkrétneho dlhujúceho, ktoré nahrádza pôvodné zobrazenie dlhov pre jednu osobu. Okno „PREVER“ slúži na zobrazenie dlhov, ktoré pridali užívatelia, ktorý patria do skupiny v ktorých je prihlásený užívateľ admin. Ako je vidieť na obrázku 41, prihlásený užívateľ má možnosť tieto dlhy potvrdiť alebo zmazať. Potvrdenie dlhu znamená, že dlh bude zviditeľnený konkrétnemu dlhujúcemu užívateľovi a zároveň mu je odoslaný dlh o novom

dlhu. Pri zmazaní dlhu je daný dlh úplne vymazaný z databáze.



Obrázek 40 Admin mód – DLHY



Obrázek 41 Admin mód - PREVER

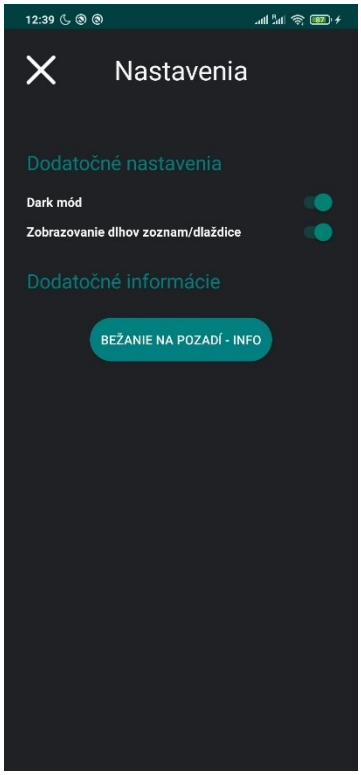
13.10 Nastavenia

Nastavenia (obrázok 42) slúžia na nastavenie toho, ako bude aplikácia vyzerať a na zobrazenie dodatočných informácií k nastaveniam, ktoré je treba zmeniť v systéme. Možnosti nastavení sú:

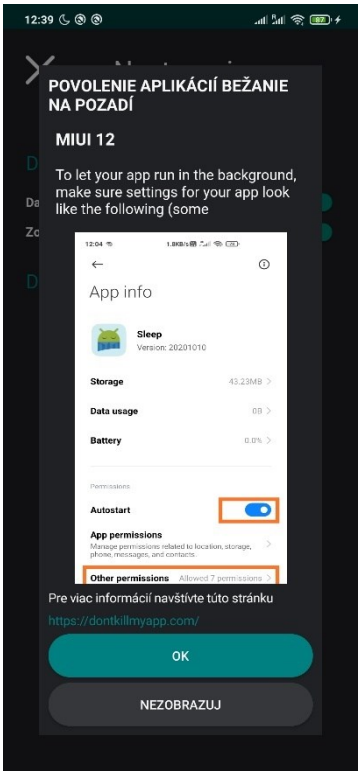
Dark mód – Slúži na zmenu témy aplikácie. Predvolená téma pri nainštalovaní aplikácie je svetlá. Pre zmenu témy na tmavú užívateľ klikne na prepínač. Taktiež klikne na prepínač pre zmenu z tmavej späť na svetlú.

Zobrazovanie dlhov zoznam/dlaždice – Slúži na zmenu zobrazovania výpisu dlhov. Predvolený režim je zoznam (obrázok 30). Pre zmenu zobrazovania výpisu dlhov na dlaždice musí užívateľ kliknúť na prepínač. Pri opätovnom kliknutí sa mód zobrazovania zmení späť na zoznam.

Dodatočné informácie v aplikácií sú ohľadom zapnutia potrebných nastavení v systéme pre fungovanie opakovaných notifikácií (12.5). Náhľad tejto stránky je na obrázku 43.



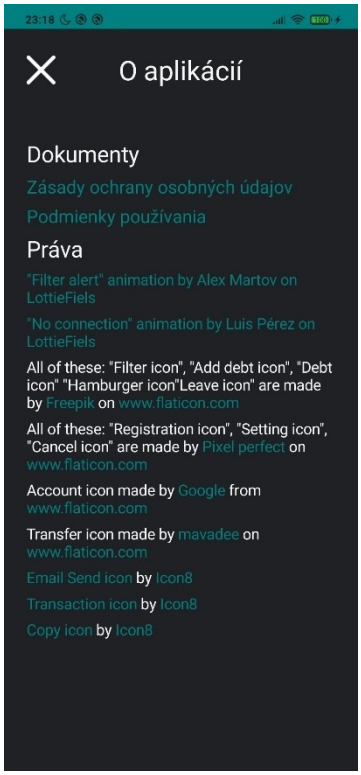
Obrázek 42 Nastavenia



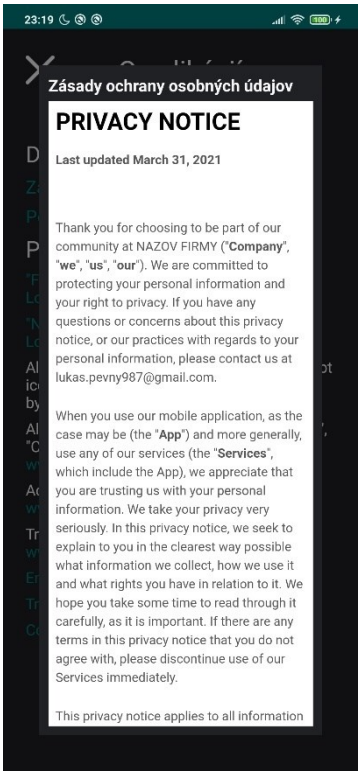
Obrázek 43 Don’t kill my app

13.11 O aplikácií

Táto stránka (obrázok 44) slúži k spätnému k informáciám o zásadách používania aplikácie a o zásadách ochrany osobných údajov (obrázok 45). Taktiež sa na tejto stránke nachádzajú zdroje k ikonám, ktoré sú v aplikácií použité.



Obrázek 44 O aplikácií



Obrázek 45 Zásady ochrany osobných údajov

ZÁVĚR

Hlavným cieľom práce bolo vytvoriť aplikáciu podľa požiadaviek pomocou frameworku Xamarin.Forms, popísať tento framework a jeho nástupcu MAUI, popísať súčasný stav vývoja mobilných aplikácií a spracovať prehľad doporučených architektúr, návrhových vzorov a techník.

V teoretickej časti sa čitateľ v krátkosti zoznámi s možnosťami vývoja mobilných aplikácií. Taktiež sa užívateľ zoznámi s operačnými systémami Android a iOS, na ktoré bola aplikácia vyvíjaná. Užívateľ by mal byť oboznámený aké programy, knižnice a aplikácie tretích strán boli pri vývoji použité a aj dôvod ich použitia. V poslednej časti sú zhrnuté odporučené architektúry, návrhové vzory a techniky, ktoré boli v aplikácií použité alebo sa využívajú pri iných typoch programovania mobilných aplikácií.

V praktickej časti sú stanovené požiadavky, ktoré mali byť splnené pri vývoji aplikácie. Myslím, že všetky tieto požiadavky sa splniť podarilo a ďalšie možné vylepšenia alebo potrebné úpravy aplikácie pre uvedenie na trh budú spomenuté nižšie. Ďalej sú v praktickej časti predstavené kľúčové prvky aplikácie formou implementácie v aplikácií z pohľadu systému. V poslednej kapitole praktickej časti sa čitateľ dočíta o všetkých funkciách aplikácie a ako s nimi pracovať.

Pre nasadenie na trh by bolo vhodné pridať určité funkcie. Najviac potrebná funkcia je podľa mňa vhodnejšie vkladanie dlhov. Toto bolo treba vyriešiť nájdením vhodnej služby na čítanie pokladničného bloku pomocou fotoaparátu. Nájdené služby ale neposkytovali údaje z daného bloku ktoré boli v aplikácií potrebné. Pre slovenský trh sa riešenie našlo, ale v čase písania práce dané API nebolo prístupné. Každý pokladničný blok na Slovensku vytlačený z ekasy má QR kód, ktorý po overení na stránke Finančnej správy vypíše všetky položky z daného bloku. Vďaka tomuto API by aplikácia mala údaje o všetkých informáciách z daného pokladničného bloku a mohli by byť spracované na potrebné účely. Ďalšie možné vylepšenia by mohli pribudnúť ohľadom splatenia dlhu. Aplikácia by mohla obsahovať platobnú bránu, poprípade iné riešenie na priamu platbu. Poprípade by mohla byť platba taktiež riešená pomocou kryptomien. Pre úspešné nasadenie by bolo určite ešte vhodné spraviť anglickú lokalizáciu aplikácie a otestovať databázu a aj aplikáciu z pohľadu výkonu pri načítavaní veľkého množstva dlhov. Vzhľadom na nejasnú funkcionalitu niektorých tlačidiel by bolo tiež správne pridať do aplikácie určitým spôsobom predstavenie funkcií aplikácie.

Pri vývoji sa vyskytlo viacero problémov frameworku Xamarin.Forms. Jeden z nich bolo ťažšie implementovanie požadovaného užívateľského rozhrania, napríklad základný element na vybratie prvku zo zoznamu neobsahuje možnosť na zmenu pozadia tohto zoznamu. Taktiež základný výber dátumu neobsahuje možnosť na zmenu farby pozadia a ostatných častí kalendára.

Na záver by bolo vhodné dodať, že aplikácia, ktorá bola v rámci tejto práci vytvorená by si po ďalšom vývoji a pridaní spomínaných funkcií mohla nájsť svoje miesto na trhu.

SEZNAM POUŽITÉ LITERATURY

[1] PEVNÝ, Lukáš. Aplikácia na správu dlhov. 2020.

[2] *Mobile Application Development* [online]. Amazon Web Services [cit. 2021-03-22]. Dostupné z: https://aws.amazon.com/mobile/mobile-application-development/

[3] *Swift* [online]. Apple developer [cit. 2021-03-22]. Dostupné z: https://developer.apple.com/swift/

[4] *Application Fundamentals* [online]. Developer Android [cit. 2021-03-22]. Dostupné z: https://developer.android.com/guide/components/fundamentals?hl=en

[5] *Most Loved, Dreaded, and Wanted Other Frameworks, Libraries, and Tools* [online]. Insights StackOverflow [cit. 2021-04-17]. Dostupné z: https://insights.stackoverflow.com/survey/2020#technology-most-loved-dreaded-and-wanted-other-frameworks-libraries-and-tools

[6] *What is Xamarin.Forms ?* [online]. Microsoft Docs, 2020 [cit. 2021-03-22]. Dostupné z: https://docs.microsoft.com/sk-sk/xamarin/get-started/what-is-xamarin-forms

[7] BENNETT, Jim. Xamarin in action: creating native cross-platform mobile apps. Shelter Island: Manning, 2018. ISBN 978-1617294389

[8] *Introducing .NET Multi-platform App UI* [online]. Microsoft Devblogs - .NET, 2020 [cit. 2021-03-22]. Dostupné z: https://devblogs.microsoft.com/dotnet/introducing-net-multi-platform-app-ui/

[9] *The New .NET Multi-platform App UI* [online]. Microsoft Devblogs - Xamarin, 2021 [cit. 2021-03-22]. Dostupné z: https://devblogs.microsoft.com/xamarin/the-new-net-multi-platform-app-ui-maui/

[10] *Xamarin.Forms - Feature Roadmap* [online]. Github - xamarin/Xamarin.Forms [cit. 2021-03-22]. Dostupné z: https://github.com/xamarin/Xamarin.Forms/wiki/Feature-Roadmap

[11] *.NET MAUI Roadmap* [online]. Github - dotnet/maui [cit. 2021-03-22]. Dostupné z: https://github.com/dotnet/maui/wiki/Roadmap

[12] *Markup Language* [online]. TechTerms, 2011 [cit. 2021-03-29]. Dostupné z: https://techterms.com/definition/markup_language

[13] *XAML Basics* [online]. Microsoft Docs, 2017 [cit. 2021-03-29]. Dostupné z: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/xaml/xaml-basics/

[14] HERMES, Dan. Building xamarin.forms mobile apps using xaml: mobile cross-platform xaml and xamarin.forms fundamentals. New York, NY: Springer Science Business Media, 2019. ISBN 978-148-4240-298.

[15] *Mobile Operating System Market Share Worldwide* [online]. Statcounter [cit. 2021-03-23]. Dostupné z: https://gs.statcounter.com/os-market-share/mobile/worldwide

[16] *Android - Overview* [online]. tutorialspoint [cit. 2021-03-23]. Dostupné z: https://www.tutorialspoint.com/android/android_overview.htm

[17] *Android - Language Breakdown* [online]. Synopsys - Open Hub [cit. 2021-03-23]. Dostupné z: https://www.openhub.net/p/android/analyses/latest/languages_summary

[18] *Android - Platform Architecture* [online]. Developer Android [cit. 2021-03-23]. Dostupné z: https://developer.android.com/guide/platform

[19] *Runtime* [online]. Stackoverflow, 2010 [cit. 2021-03-23]. Dostupné z: https://stackoverflow.com/questions/3900549/what-is-runtime/3900561

[20] *Overview of iOS Platform* [online]. Echo inovative IT [cit. 2021-03-23]. Dostupné z: https://echoinnovateit.com/ios-platform-overview/

[21] MICROSOFT CORPORATION. *Microsoft Visual Studio Community 2019, Version 16.9.1* [software]. [prístup 24.03.2021] Dostupné z: https://visualstudio.microsoft.com/downloads/. Požiadavky na system: Windows 10 version 1703 or higher: Home, Professional, Education, and Enterprise (LTSC and S are not supported); Windows Server 2019: Standard and Datacenter; Windows Server 2016: Standard and Datacenter, Windows 8.1 (with Update 2919355): Core, Professional, and Enterprise; Windows Server 2012 R2 (with Update 2919355): Essentials, Standard, Datacenter; Windows 7 SP1 (with latest Windows Updates): Home Premium, Professional, Enterprise, Ultimate; 1.8 GHz or faster processor. Quad-core or better recommended 2 GB of RAM; 8 GB of RAM recommended; minimálne 800MB miesta na disku

[22] *Visual Studio IDE - Overview* [online]. Microsoft Docs, 2019 [cit. 2021-03-24]. Dostupné z: https://docs.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2019

[23] *AWS SDK for .NET* [online]. Github - aws/aws-sdk-net [cit. 2021-03-26]. Dostupné z: https://github.com/aws/aws-sdk-net/

[24] *Lottie Xamarin* [online]. Github - Baseflow/LottieXamarin [cit. 2021-03-26]. Dostupné z: https://github.com/Baseflow/LottieXamarin

[25] *Push notifications - explained* [online]. AIRSHIP [cit. 2021-03-26]. Dostupné z: https://www.airship.com/resources/explainer/push-notifications-explained/

[26] *FirebaseNet* [online]. Github - tiagomtotti/firebaseNet [cit. 2021-03-26]. Dostupné z: https://github.com/tiagomtotti/firebaseNet

[27] *Html Agility Pack* [online]. Html Agility Pack [cit. 2021-03-26]. Dostupné z: https://html-agility-pack.net/

[28] *Monkey Cache* [online]. Github - jamesmontemagno/monkey-cache [cit. 2021-03-26]. Dostupné z: https://github.com/jamesmontemagno/monkey-cache

[29] *Firebase Push Notification Plugin* [online]. Github - CrossGeeks/FirebasePushNotificationPlugin [cit. 2021-03-26]. Dostupné z: https://github.com/CrossGeeks/FirebasePushNotificationPlugin

[30] *Popup Page Plugin* [online]. Github - rotorgames/Rg.Plugins.Popup [cit. 2021-03-26]. Dostupné z: https://github.com/rotorgames/Rg.Plugins.Popup

[31] *Local Notifications Plugin* [online]. Github - edsnider/localnotificationsplugin [cit. 2021-03-26]. Dostupné z: https://github.com/edsnider/LocalNotificationsPlugin

[32] *Xamarin Community Toolkit* [online]. Github - xamarin/XamarinCommunityToolkit [cit. 2021-03-28]. Dostupné z: https://github.com/xamarin/XamarinCommunityToolkit

[33] *Xamarin.Forms 5 Is Here! | The Xamarin Show* [online]. Youtube [cit. 2021-03-28]. Dostupné z: https://youtu.be/ttF80UnrJAg. Kanál používateľa Xamarin.Developers

[34] *Xamarin.Essentials* [online]. Github - xamarin/Essentials [cit. 2021-03-28]. Dostupné z: https://github.com/xamarin/Essentials

[35] *Xamarin.Essentials - Feature Guides* [online]. Microsoft Docs [cit. 2021-03-28]. Dostupné z: https://docs.microsoft.com/sk-sk/xamarin/essentials/?WT.mc_id=friends-0000-jamont

[36] *FFImageLoading* [online]. Github - luberda-molinet/FFImageLoading [cit. 2021-03-28]. Dostupné z: https://github.com/luberda-molinet/FFImageLoading

[37] *Xamarin.Firebase.Config* [online]. Github - xamarin/GooglePlayServicesComponents [cit. 2021-03-28]. Dostupné z: https://github.com/xamarin/GooglePlayServicesComponents

[38] *Xamarin.Forms.PancakeView* [online]. Github - sthewissen/Xamarin.Forms.PancakeView [cit. 2021-03-28]. Dostupné z: https://github.com/sthewissen/Xamarin.Forms.PancakeView

[39] *Hello World* [online]. Github Guides [cit. 2021-03-29]. Dostupné z: https://guides.github.com/activities/hello-world/

[40] *Commit vs pull request* [online]. StackOverflow, 2016 [cit. 2021-03-29]. Dostupné z: https://stackoverflow.com/questions/35007939/what-is-the-difference-between-commits-and-pull-requests

[41] *Overview of Amazon Web Services* [online]. AWS Docs, 2020 [cit. 2021-03-28]. Dostupné z: https://docs.aws.amazon.com/whitepapers/latest/aws-overview/introduction.html

[42] *Encryption - how it works* [online]. AWS docs [cit. 2021-04-15]. Dostupné z: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/encryption.howitworks.html

[43] *Encryption - usage notes* [online]. AWS docs [cit. 2021-04-15]. Dostupné z: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/encryption.usagenotes.html

[44] *Firebase Pricing* [online]. Firebase [cit. 2021-04-04]. Dostupné z: https://firebase.google.com/pricing

[45] *Firebase Overview* [online]. Firebase [cit. 2021-04-04]. Dostupné z: https://firebase.google.com/

[46] *Firebase - Build Products* [online]. Firebase - Products / Build [cit. 2021-04-04]. Dostupné z: https://firebase.google.com/products-build

[47] *Firebase - Release Products* [online]. Firebase - Products / Build [cit. 2021-04-04]. Dostupné z: https://firebase.google.com/products-release

[48] *Firebase - Release Products* [online]. Firebase - Products / Release & Monitor [cit. 2021-04-04]. Dostupné z: https://firebase.google.com/products-release

[49] *Spoznajte payme* [online]. payme [cit. 2021-04-15]. Dostupné z: https://www.payme.sk/spoznajte-payme

[50] *Our mission* [online]. Don't kill my app! [cit. 2021-04-15]. Dostupné z: https://dontkillmyapp.com/problem

[51] *MVVM Pattern Using the Prism Library* [online]. Microsoft Docs, 2014 [cit. 2021-03-30]. Dostupné z: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/gg405484(v=pandp.40)

[52] *The Model-View-ViewModel Pattern* [online]. Microsoft Docs, 2017 [cit. 2021-03-30]. Dostupné z: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm

[53] *MVC Tutorial for Beginners* [online]. Guru99 [cit. 2021-04-18]. Dostupné z: https://www.guru99.com/mvc-tutorial.html

[54] *Dependency Injection* [online]. TutorialsTeacher [cit. 2021-04-18]. Dostupné z: https://www.tutorialsteacher.com/ioc/dependency-injection

[55] *IoC Container* [online]. TutorialsTeacher [cit. 2021-04-18]. Dostupné z: https://www.tutorialsteacher.com/ioc/ioc-container

[56] *Singleton Design Pattern | Introduction* [online]. GeeksforGeeks [cit. 2021-03-31]. Dostupné z: https://www.geeksforgeeks.org/singleton-design-pattern-introduction/

[57] *Implementing the Singleton Pattern in C#* [online]. C# in Depth, 2011 [cit. 2021-03-31]. Dostupné z: https://csharpindepth.com/articles/singleton

[58] *DependencyService Introduction* [online]. Microsoft Docs, 2019 [cit. 2021-04-03]. Dostupné z: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/dependency-service/introduction

[59] *.NET: Object Persistence Model* [online]. AWS Docs [cit. 2021-04-06]. Dostupné z: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DotNetSDKHighLevel.html

[60] *DynamoDB Scan vs Query* [online]. Dynobase [cit. 2021-04-07]. Dostupné z: https://dynobase.dev/dynamodb-scan-vs-query/

[61] *Rfc2898DeriveBytes Class* [online]. Microsoft Docs [cit. 2021-5-3]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.rfc2898derivebytes?view=net-5.0

[62] *What is NoSQL Injection and How Can You Prevent It?* [online]. Netsparket, 2020 [cit. 2021-5-4]. Dostupné z: https://www.netsparker.com/blog/web-security/what-is-nosql-injection/

[63] *About FCM messages* [online]. Firebase [cit. 2021-04-08]. Dostupné z: https://firebase.google.com/docs/cloud-messaging/concept-options

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API	Application programmable interface (rozhranie pre programovanie aplikácií)
XAML	eXtensible Markup Language
HTML	Hyper Text Markup Language
CRUD	Create, read, update, delete
IBAN	International Bank Account Number

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

PŘÍLOHA P I: CD-ROM

Priložené CD obsahuje:

- Zdrojové kódy aplikácie vo formáte .zip
- Bakalársku prácu vo formáte .pdf