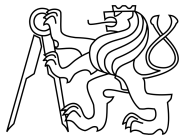
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Vývoj multiplatformní mobilní aplikace pro správu elektronických receptů
Student:	Bc. Marek Roubíček
Vedoucí:	Ing. Petr Oliva, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

1. Seznamte se s možnostmi vývoje mobilních aplikací s ohledem na požadavek multiplatformního využití aplikace.
2. Popište standardní metodiky vývoje pro platformy Android a iOS.
3. Popište metodiky multiplatformního vývoje ve vybrané technologii z bodu 1.
4. Společně s vedoucím práce specifikujte požadavky na vlastní mobilní aplikaci pro zpracování elektronických receptů pro Státní ústav pro kontrolu léčiv.
5. Implementujte klientské aplikace pro pacienty a lékaře na vybrané platformě na základě požadavků v bodě 4.
6. Publikujte hotové mobilní aplikace na oficiální úložiště aplikací.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 7. února 2017

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra Softwarového inženýrství

Diplomová práce

Multiplatformní mobilní aplikace pro správu elektronických receptů

Bc. Marek Roubíček

Vedoucí práce: Ing. Petr Oliva, Ph.D.

8. ledna 2018

Poděkování

Rád bych poděkoval svojí rodině a blízkým za morální podporu při psaní této práce. Děkuji svému zaměstnavateli společnosti Solitea Business Solutions s.r.o. za možnost podílet se na projektu eRecept. Dále bych rád poděkoval svému vedoucímu práce, který mi poskytl konzultace k práci. V neposlední řadě chci poděkovat všem uživatelům výsledných aplikací za jejich využívání v reálném provozu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. Dále prohlašuji, že jsem s Českým vysokým učením technickým v Praze uzavřel dohodu, na základě níž se ČVUT vzdalo práva na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona. Tato skutečnost nemá vliv na ust. § 47b zákona č. 111/1998 Sb., o vysokých školách, ve znění pozdějších předpisů.

V Praze dne 8. ledna 2018 .....

Odkaz na tuto práci

Roubíček, Marek. *Multiplatformní mobilní aplikace pro správu elektronických receptů.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Práce se zabývá nástroji pro multiplatformní vývoj mobilních aplikací, které díky znalosti jedné technologie umožňují vyvíjet aplikace na více platforem současně. Dále jsou vysvětleny principy technologie Xamarin, použité pro vývoj vlastních aplikací pro správu elektronických receptů pacienty a lékaři. V této práci je popsán koncept elektronických receptů, vývoj samotných aplikací a jejich nasazení na obchody Google Play a App Store.

Klíčová slova Mobilní aplikace, Multiplatformní vývoj, Xamarin, C#, XAML, Android, iOS, MVVM, Státní ústav pro kontrolu léčiv, Elektronizace státní správy, eRecept, Publikování na Google Play a App Store

Abstract

The thesis explores tools for multiplatform mobile application development, providing parallel app development on multiple platforms with one knowledge base. There are described principals of Xamarin technology, which is further used for development of application for electronic prescription management by doctors and patients. This thesis describes concept of ePrescription, development workflow and publishing of applications to Google Play and App Store stores.

Keywords Mobile app, Multiplatform development, Xamarin, C#, XAML, Android, iOS, MVVM, State Institute for Drug Control, eGovernment, ePrescription, Publishing on Google Play and App Store

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Vedle přesunu informačních systémů do cloudu je používání mobilních aplikací fenoménem dnešní doby. Stále více vývojářských firem se zaměřuje na vývoj pro telefony a jiná mobilní zařízení, neboť se jedná o velmi rozmanitý trh s obrovským potenciálem. Jen v roce 2015 uživatelé uskutečnili 156 miliard instalací nejrůznějších aplikací a do roku 2020 je odhadovaný nárůst na 200 miliard instalací[1]. Je tedy správné věnovat své zdroje na vývoj mobilních aplikací.

V současné době jsou nejrozšířenějšími mobilními platformami systémy Android a iOS (98% podíl na trhu)[2]. K dispozici je stále více různých přístrojů, které běží na těchto systémech a udržovat podporu co největšího počtu zařízení je stále složitější neboť společnosti vyrábějící přenosná zařízení se předhánějí v inovacích a integraci co největší funkcionality do svých přístrojů. Jak tedy obsáhnout rostoucí možnosti a požadavky uživatelů? Lze se vypořádat s požadavky na výkon a maximální využití nativních funkcí nějakým obecným řešením?

Velmi malý podíl aplikací se vyvíjí s úmyslem být podporovány pouze pro jednu ze zmíněných platforem - jedná se především o systémové utility, možnosti personalizace systému a podobně. Daleko větší zastoupení mají dohromady všechny ostatní kategorie, které se zaměřují na obecné použití mobilních telefonů (Vzdělávání, sociální média, životní styl, bussinesové aplikace, hudba, knihy, cestování atd. [3]). Standardem v oblasti aplikací je tedy podpora pro obě majoritní platformy. Je v současné době možné zefektivnit vývoj a vyvíjet takové aplikace se sdíleným kódem?

Česko je až padesáté na světě v hodnocení vyspělosti takzvaného eGovernmentu [4], z tohoto pohledu má Česká republika na čem pracovat. Vedle zavádění systémů pro státní správu, které mají ulehčit administrativní zatížení státu se nabízí možnost využití moderních technologií, a tedy logicky i mobilních aplikací ku prospěchu občanů České republiky. Jednou z oblastí pro veřejný sektor je zavádění takzvaného elektronického receptu, který pomůže uživatelům sledovat, spravovat a mít pod kontrolou předpisy na své léky.

V této práci je použitá celá řada obrázků popisujícími různá schemata (převážně technologie Xamarin). Jelikož se na obrázcích vyskytuje řada odborných pojmů, nepřeložitelných do českého jazyka, jsou obrázky ponechány v původním jazyce včetně případných doplňujících popisků.

Cíle práce

Multiplatformní vývoj mobilních aplikací je moderní koncept vývoje, který umožňuje sdílení většiny zdrojového kódu bez ohledu na cílovou platformu aplikace. Cílem této práce je prozkoumat možnosti multiplatformního mobilního vývoje a aplikovat poznatky při vytvoření vlastních mobilních aplikací pro Státní ústav pro kontrolu léčiv, které budou sloužit pro správu a vytváření elektronických receptů lékařům a veřejnosti. Vyvinuté aplikace budou vystaveny na internetových obchodech s aplikacemi Google Play a App Store.

Část I

Teoretická část

Kapitola 1

Specifika mobilních aplikací

Mobilní aplikace tak, jak je známe dnes, plní různorodé úlohy napříč informačními systémy. Jedná se o způsob, jak dostat nějakou funkcionalitu co nejblíže koncovému uživateli. Mobilní aplikace mají pro uživatele svá specifika a omezení, kterým je potřeba se přizpůsobit. Navíc se dají při vývoji využít technologie, které jsou na běžných počítačích těžko použitelné nebo naprosto nedostupné.

Problém vývoje mobilních aplikací je i rychlé zastarávání mobilních technologií – morální životnost mobilních telefonů a obecně mobilních zařízení je zhruba rok, poté již funkčně a výkonově zaostávají. Následně se pak musejí aktualizovat i mobilní aplikace.

1.1 Mobilita

Klíčová vlastnost používání mobilních zařízení je možnost přenositelnosti. Aplikace na různých tabletech, phabletech, mobilních telefonech nebo chytrých hodinkách se dají použít i v místech, kde není dostupná elektrická síť. To s sebou přináší i jistá úskalí, jako například potřebu myslet na omezenou kapacitu baterie. To u systémů, které jsou koncipovány pro nepřetržitý provoz, může být v podobě mobilní aplikace nepraktické.

1.2 Zobrazovací prostor

Zobrazovací prostor, tzv. **Form factor** je pojem, který udává velikost displeje, na který lze informace zobrazovat. S tím je spojený požadavek na aplikace, aby byly přehledné a zobrazovaly tak co nejméně informací současně. Ve výsledku se aplikace pro mobilní zařízení mohou částečně nebo i zcela lišit na jednotlivých platformách. Důležité je také myslet na to, že každé zařízení, na kterém aplikace poběží má jiný form factor, a informace by měly být přizpůsobené cílovému zařízení. Toho dnes již nelze dosáhnout rozsáhlým stromem podmínek pro každé zařízení zvlášť, neboť různých zařízení jsou již tisíce, proto by měly aplikace splňovat alespoň částečnou responzivitu.

1.3 Speciální funkce

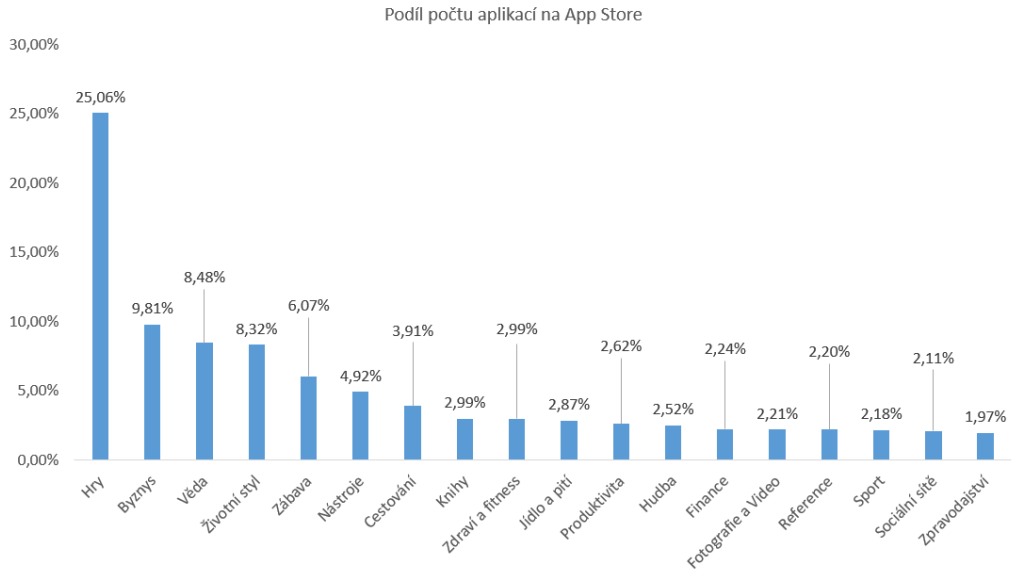
Mobilní zařízení nabízí velkou množinu speciálních funkcí, které lze při vývoji aplikací využívat:

- Gyroskop
- Fotoaparát
- GPS
- Bluetooth
- GSM
- atd.

Dobrá aplikace využívá jen ty funkce, které pomáhají k jejímu cíli. Ostatní podružné úlohy je dobré přenechat na základním programovém vybavení zařízení anebo dalším specializovaným aplikacím. Nemá smysl se například zabývat zobrazováním PDF dokumentů u aplikace sloužící pro deníkové zápisy.

1.4 Typologie aplikací

Mobilní aplikace lze rozdělit do kategorií podle typu jejich využití. Podle serveru statista.com bylo v říjnu 2017 na internetovém obchodu App Store společnosti Apple k dispozici 2,2 milionu aplikací ke stažení. Na grafu podílu počtu aplikací na AppStore k říjnu 2017 1.1 [5] je vidět distribuce jednotlivých kategorií. Rozložení jednotlivých kategorií se v procentech může lišit na jednotlivých platformách, nicméně škála použití aplikací je v zásadě shodná.



Obrázek 1.1: Podíl počtu aplikací na App Store k říjnu

Kapitola 2

Volba přístupu pro vývoj multiplatformní aplikace

Možností jak vyvíjet aplikace je pro každou platformu mnoho. Spolu se zveřejněním API pro vývoj aplikací na jednotlivých platformách vznikla postupem času spousta nástrojů, frameworků nebo běhových prostředí, které umožňují psát aplikace i v jiných jazycích, než je hlavní jazyk platformy.

Každá strategie má svá specifika a svoje využití, proto je volba přístupu na začátku tvorby aplikace to nejdůležitější rozhodnutí. Pro účely této práce jsou uvedeny k porovnání pouze ty přístupy, které se dají souhrnně nazvat multiplatformní, tedy takový vývoj aplikací, který zajistí, že vyvinutá aplikace poběží na minimálně dvou rozdílných platformách.

V následujících kapitolách je porovnání jednotlivých přístupů vývoje aplikací z pohledu jejich hlavních rozdílů, výhod a nevýhod. Dále existuje již celá řada benchmarků pro testování výkonu, které se zaměřují na porovnání rychlostí jednotlivých technologií při běhu na cílovém zařízení při konkrétních úkolech a procesech. Pro větší názornost používám benchmark ze serveru Magenic[6], který porovnává aplikace napsané různými přístupy z hlediska jejich náročnosti na spotřebu zdrojů (výpočetní, paměťové).

2.1 Přístupy vývoje

2.1.1 Nativní vývoj

Primárním přístupem pro vývoj aplikací je programování v nativním jazyce dané platformy. Přesněji řečeno jedná se o používání vystaveného API dané platformy, což je pro operační systém Android jazyk Java, pro operační systém iOS jazyk Objective-C (s nadstavbou Swift) a pro operační systém Windows phone jazyk C#. Pokud si chce dnes firma nechat vyvinout mobilní aplikaci pro tři hlavní platformy (Android, iOS, Windows Phone) v nativním jazyce, nezbývá ji, než sehnat 3 vývojáře, kteří budou implementovat stejnou věc 3x jinak, aby docílili stejného výsledku[7]. To samozřejmě přináší starosti na straně vstupních nákladů i budoucího rozvoje vyvinutých systémů.

2.1.1.1 Výhody nativního vývoje

- **Výkon** - Aplikace jsou napsány ve více „low-level“ jazycích než hybridní aplikace nebo mobilní web.
- **Podpora nejnovějších API** - Lze používat nespočet různých API telefonu.
- **Nativní UI prvky** - Vývojář software podporuje UI prvky přímo v telefonu.

2.1.1.2 Nevýhody nativního vývoje

- **Vysoká cena** - Aplikaci je nutné napsat pro každou platformu znovu, takže výsledná aplikace je násobně dražší.
- **Nároky na programátory** - Je nutné mít programátory, kteří umí hned několik zcela syntakticky odlišných jazyků: Objective C/Swift, Java a C#/C++.

- **Náročné opravy aplikací** - Stejné úpravy v aplikaci se musejí provádět v každé aplikaci zvlášť a při změnách verzí každého z operačních systémů, které nejsou nijak časově synchronizovány.

2.1.2 Hybridní vývoj

Další možností vývoje aplikace pro více platforem je využití frameworků. Aby se předešlo výše uvedeným nevýhodám paralelního vývoje téže aplikace pro více platforem současně, tři zmíněné nejpoužívanější platformy mají ve svém API k dispozici tzv. **WebView**, což je jednoduchá instance webového prohlížeče, která se dá použít jako ovládací prvek zabudovaný v aplikaci. Jedná se o podobný princip, jako jsou iframe v jazyce HTML pro webové stránky, kdy obsah daného ovládacího prvku nespravuje přímo aplikace, ale služba na pozadí. Celý vývoj grafického prostředí pak probíhá v jazycích HTML a CSS a logika je řešena JavaScriptem. Mezi zástupce hybridního přístupu lze řadit následující frameworky:

- Cordova (Phone Gap)
- Ionic
- Framework7
- Onsen

2.1.2.1 Výhody hybridního vývoje

- **Multiplatformnost** - Lze vyvíjet aplikaci 2-3x levněji než při vývoji nativních aplikací.
- **Snadné testování** - 90 % práce na aplikaci lze simulovat v prohlížeči. Není potřeba zdlouhavě nasazovat aplikaci na zařízení nebo do simulátoru.
- **Webové technologie** - Vývoj aplikace vyžaduje znalosti webových technologií, které jsou zpravidla jednodušší na naučení než nativně používané jazyky (Java, ObjectiveC/Swift, C#/C++).
- **Jednodušší update při změnách OS** - Provádí se pouze při novém vydání verze vybraného frameworku.

2.1.2.2 Neýhody hybridního vývoje

Hlavní problém těchto řešení je nedostupnost funkcionalit mobilního zařízení (např. fotoaparátu nebo GPS) z webového prohlížeče. Kvůli tomu lze ke zmíněným frameworkům doinstalovat na úrovni vývoje pluginy, které zabalují aktivní API pro použití v již zmíněném ovládacím prvku WebView. Pomocí speciálních příkazů JavaScriptu se pak dají zabalené funkce vyvolávat. Hlavní nevýhody jsou:

- **Vyšší požadavky na hardware** - Vzhledem k tomu, že webová aplikace, která se tváří jako mobilní aplikace je ve skutečnosti jen vložena v instanci prohlížeče, tak to znamená jistá hardwarová omezení. V dnešní době se tento problém minimalizuje se zlepšujícím se výkonem telefonů a prohlížečů.
- **Možná nedostupnost API** - Vývojáři neustále vyvíjejí nová API, pro přístup k novým funkcím telefonu. V případě hybridní aplikace se může stát, že nově dostupná API nejsou podporována (drtivá většina používaných však je).
- **Chybí nativní UI prvky** - Velkou nevýhodou je to, že zde v základu naprosto chybí jakékoliv UI prvky. Existující frameworky již ovládací prvky s nativním vzhledem mají.

2.1.3 Mobilní responzivní web

V mnoha případech je aplikace pro mobilní zařízení co do použitelnosti a funkcionality věrnou kopií již existující webové aplikace. V takovém případě nemá smysl vyvíjet aplikaci zcela novou, ale stačí pouze přizpůsobit vizuální podobu webu pro menší, specifické displeje mobilních zařízení. Jak již bylo zmíněno v kapitole 1.2, řešením je responzivní design, který pokryje různé velikosti displejů. Nevýhodou v takovém případě může být distribuce takové aplikace, kdy je vlastně potřeba

distribuovat URL adresu daného webu. Ačkoliv existuje možnost vytvořit si odkaz na web pomocí ikonky na ploše mobilního zařízení, automatické vytvoření ikonky při instalaci mobilní aplikace je více uživatelsky přívětivá.

2.1.3.1 Výhody mobilního webu

- **Rychlost vývoje** - Vývoj mobilního webu v případě jednodušších webů zahrnuje pouze změnu CSS.
- **Univerzálnost** - Při úpravách na webu není potřeba měnit kód ve více aplikacích.
- **Není nutné instalovat aplikaci do telefonu.**
- **Aplikace není potřeba distribuovat přes obchod třetích stran** - Veškeré změny v aplikaci se projeví okamžitě po nasazení bez nutnosti stažení nové verze zákazníkem.

2.1.3.2 Nevýhody mobilního webu

- **Uživatelský prožitek** - Je mnohem složitější přizpůsobit web bez dodatečných knihoven tak, aby uživatel měl dojem z webu stejný, jako v případě používání nativní aplikace.
- **Chybí napojení na API telefonu** - Některé služby jako například GPS je možné napojit i do webové aplikace, ale proti nativnímu nebo hybridnímu vývoji jsou možnosti chudší.
- **Chybí nativní prvky UI** - Vše je potřeba si nastylovat, nebo zkusit najít knihovny, které tyto prvky implementují. Často je složité je implementovat na již existující web, protože mohou měnit strukturu a je potřeba pamatovat na to, jak vypadá web i v počítači.
- **Nutnost být on-line a mít otevřený web v prohlížeči** - Ještě stále každý nemá připojení k internetu, nebo má omezený limit stahovaných dat.

2.1.4 Grafické frameworky

Další z možností tvorby mobilních aplikací jsou specializované frameworky, které využívají již existujících knihoven a API pro práci s 2D a 3D grafikou. Hodí se zejména pro vytváření různých simulací nebo tvorbu her. Pomocí vlastního API vytváří jakýsi adaptér, kterému stačí při sestavování aplikace nakonfigurovat cílovou platformu, na které bude aplikace běžet a kompilátor se postará o sestavení potřebné struktury. Mezi nejrozšířenější frameworky tohoto typu lze řadit:

- **Unity** - Pro mobilní aplikace používá sadu knihoven OpenGL dostupnou jak pro Android tak pro iOS. Výhodou Unity je, že má vlastní vývojové IDE, ve kterém se dají generovat 2D i 3D modely. Mobilní zařízení jsou pouze podmnožinou platforem, na které lze pomocí Unity vyvíjet. Pod sdíleným kódem se například dají vyvíjet i aplikace na herní konzole[8].
- **Qt** - Framework, který je postaven nad jazykem C++, do kterého přináší řadu dalších rozšíření. Dovoluje vytvářet 2D i 3D aplikace také pomocí knihoven OpenGL. Pro grafické rozhraní definuje vlastní jazyk QML[9], kterým lze vytvářet ovládací prvky typu tlačítko apod.

2.1.4.1 Výhody grafických frameworků

- **Univerzálnost** - Aplikace mohou běžet na velkém počtu naprosto rozdílných, nejen mobilních platforem.
- **3D grafika** - Možnost vytvářet složité grafické modely pomocí specializovaných knihoven.
- **Vlastní intuitivní IDE** - Pro obě zmíněné technologie existuje vlastní, intuitivní IDE, pomocí kterého lze snadno vytvářet scény a objekty pro 3D grafiku.

2.1.4.2 Nevýhody grafických frameworků

- **Licence** - Pro použití veškeré funkcionality se platí poměrně velký licenční poplatek.
- **Příliš obsáhlé pro běžné aplikace** - Běžné aplikace nepotřebují pracovat s 3D knihovnami a výsledná aplikace je objemnější.

2.1.5 Binding - Překlad mezi programovacími jazyky

Jako poslední možnost pro vývoj pro více platforem současně je tzv. **binding**. Jedná se o automatizovaný proces, který vezme zdrojový kód napsaný v jednom jazyce a přeloží ho do jazyka cílové platformy. Díky tomu je výsledná aplikace opravdu autentická jako v případě nativního vývoje. Mezi technologie používající tento přístup patří:

- **Reactive Native**
- **Native Script**
- **Xamarin**

První dva zmíněné příklady technologií používají opět modifikovaný JavaScript a CSS, které ale nezobrazují ve WebView controlu, jako v případě Hybridních aplikací, ale překládají prvky uživatelského rozhraní na nativní elementy. Zcela jiný přístup je používán v Xamarinu, kde se za běhu používá obousměrný můstek mezi dvěma runtime prostředími, kde jednou cestou jsou volány nativní funkce a na druhé straně jsou implementována rozhraní standardního API. Více o Xamarinu v kapitole 3.

2.1.5.1 Výhody bindingu

- **Univerzálnost** - Aplikace mohou běžet na velkém počtu naprosto rozdílných, nejen mobilních platforem.
- **Jedna technologie** - Kód pro všechny platformy má stejnou syntaxi, a používá jednotné IDE.
- **Výkon téměř shodný s nativním vývojem** - Výkon je jen nerozeznatelně nižší než u nativních aplikací (nadbytečná je správa nového kódu na cílové platformě[10]).
- **Shodná uživatelská zkušenost** - Aplikace psané pomocí bindingu splňují všechny standardy cílové platformy.
- **Plná hardwarová podpora** - Lze využívat všechny funkcionality mobilního zařízení.
- **Kompletní ekosystém vývoje** - U Xamarinu lze všechno dělat v rámci jednoho IDE, vše v návaznosti (vývoj, verzování, testování, nasazování apod.).
- **Open source technologie s korporátní podporou** - Microsoft vydává pro Xamarin každý měsíc nové úpravy, opravy a doplňuje funkcionalitu.

2.1.5.2 Nevýhody bindingu

- **Pomalejší náběh aplikace** - Při startu aplikace se musí inicializovat knihovny daného frameworku, které spolupracují s nativním API (týká se jak Java Scriptu tak běhového prostředí Xamarinu).
- **Mírně zpožděná podpora nových funkcí** - Xamarin resp. Facebook (stojící za vývojem React Script) nasazují nové verze frameworku poměrně často, nicméně nové API musí být zahrnuto do nové verze.
- **Omezený přístup k open-source knihovnám** - Open source knihovny pro některé funkcionality nejsou tak rozšířené jako knihovny pro nativní vývoj (Pro nové technologie nemusí vlastní knihovna ani existovat).
- **Kompatibilita s nástroji třetích stran** - Externí utility jako hardwarové čtečky apod. jsou optimalizovány pro nativní vývoj, a nemusí pro ně být vytvořená podpora pro technologii bindingu.
- **Větší velikost výsledného balíčku** - Pro Xamarin je potřeba dát vlastní runtime jako součást výsledného balíčku, což zvětší jeho počáteční velikost o jednotky MB.

2.2 Benchmarky - srovnání nativních a hybridních aplikací

Výhody a nevýhody z pohledu užití jednotlivých přístupů byly popsány v kapitole 2.1. Dle výkonu existuje již celá řada benchmarků, které se zaměřují na porovnání rychlostí jednotlivých technologií při běhu na cílovém zařízení při konkrétních úkolech a procesech. V této práci využívám dat z testů serveru magetic.com z roku 2015 (viz. kapitola 2), kde je popsána i veškerá metodika pro porovnání. V tomto testu byly porovnávány aplikace, které měly stejný úkol (Načítání dat ze vzdáleného serveru a výpočet prvočísel do určitého limitu), a byly napsány v:

- Nativním jazyku (Java a Objective-C 64 bit)
- Hybridním frameworku Cordova (JavaScript)
- Xamarin (Technologie MonoTouch a Mono for Android)
- Xamarin.Forms

2.2.1 Velikost aplikace

Velikost aplikace byla největší pro technologii Xamarin (viz. tabulky 2.1 a 2.2) – zejména proto, že si s sebou aplikace nese zabalené knihovny pro práci s Mono runtime a v případě Xamarin.Forms ještě knihovny .NET. V době provádění testu byl Xamarin v intenzivním vývoji. Momentálně je možné jednotlivé knihovny při kompilaci tzv. linkovat (z původní SDK nechat v aplikaci pouze ty, které výsledná aplikace skutečně využije – což je ve většině případů ušetření desítek procent výsledné velikosti.)

Technologie	**Velikost**
Java	166 KB
Cordova	433 KB
Xamarin.Android	3,5 MB
Xamarin.Forms	4,7 MB

Tabulka 2.1: Porovnání velikostí aplikací Android

Technologie	**Velikost**
Objective-C(64 bit)	644 kB
Cordova	2,7 MB
Xamarin.iOS	12,1 MB
Xamarin.Forms	16,9 MB

Tabulka 2.2: Porovnání velikostí aplikací iOS

2.2.2 Načtení úvodní obrazovky

Načítání aplikace do paměti a zobrazení úvodní obrazovky je další test, který byl zmíněným pokusem prováděn. Jak je popsáno v kapitole 3.2, Xamarin potřebuje ke svému správnému fungování spustit případně navíc zkompilovat určité části Mono běhového prostředí, proto je zavedení aplikace pomalejší než u nativních aplikací. Na druhou stranu se autoři Xamarinu snaží o co nejlepší optimalizace, aby se časově i v tomto faktoru byly schopni vyrovnat nativnímu vývoji. Doba načítání na jednotlivých platformách je v tabulkách 2.3 resp. 2.4.

Technologie	**Průměrná doba načtení**
Java	1,085 s
Cordova	3,978 s
Xamarin.Android	1,704 s
Xamarin.Forms	2,764 s

Tabulka 2.3: Porovnání doby načtení aplikací Android

Technologie	Průměrná doba načtení
Objective-C(64 bit)	1,221 s
Cordova	1,715 s
Xamarin.iOS	1,28 s
Xamarin.Forms	1,813 s

Tabulka 2.4: Porovnání doby načtení aplikací iOS

2.2.3 Výpočet prvočísel

Výpočet prvočíselnosti je výpočetní problém pro brute-force algoritmy, na kterých lze zjistit, jaké je vytížení procesoru na jednotlivých platformách při základních matematických operacích. Z tohoto pohledu ze zmíněného testu vyplývá, že běžné operace jsou při vývoji v Xamarinu (.NETu) srovnatelně časově náročné jako nativní kód. Naproti tomu JavaScript byl v tomto testu mnohonásobně pomalejší viz. tabulky 2.5 resp. 2.6.

Technologie	Průměrná doba výpočtu
Java	4,342 s
Cordova	94,151 s
Xamarin.Android	4,258 s s
Xamarin.Forms	4,259 s

Tabulka 2.5: Porovnání doby výpočtu prvočísel menších než 5 000 000 aplikací Android

Technologie	Průměrná doba výpočtu
Objective-C(64 bit)	5,014 s s
Cordova	67,291 s
Xamarin.iOS	4,349 s s
Xamarin.Forms	4,379 s s

Tabulka 2.6: Porovnání doby výpočtu prvočísel menších než 5 000 000 aplikací iOS

2.2.4 Práce s databází

Pro práci s databázemi byla vytvořena autorem testu jiná aplikace, tudíž její velikost a doba načtení se lišila od původní aplikace. Přehled těchto údajů lze najít na kompletním protokolu dalšího testu na serveru Magenic[11], stejně jako použité implementace knihoven, pro práci s databázemi na jednotlivých platformách a technologiích. Výsledné časy viz. tabulky 2.7 resp. 2.8.

Technologie	Průměrná doba vkládání	Průměrná doba dotazů
Java	18,569 s	0,551 s
Cordova	24,126 s	1,117 s
Xamarin.Android	32,55 s	0,601 s
Xamarin.Forms	30,873 s	0,89 s

Tabulka 2.7: Porovnání práce s databází (Průměrná doba nad 1000 záznamy) aplikací Android

Technologie	Průměrná doba načtení	
Objective-C(64 bit)	8,044 s	0,792 s
Cordova	14,944 s	14,944 s
Xamarin.iOS	8,151 s	0,799 s
Xamarin.Forms	8,137 s	0,735 s

Tabulka 2.8: Porovnání práce s databází (Průměrná doba nad 1000 záznamy) aplikací iOS

2.3 Shrnutí

Z výše uvedených dat vyplývá, že největším rozdílem aplikace psané v Xamarin a nativním způsobem je velkost výsledné aplikace, která se ale s nabalování nové funkcionality neliší multiplikativně, nýbrž pouze počáteční velikostí. Při každém dalším přidávání funkčnosti roste výsledný rozdíl velikostí přímo úměrně. Výkonově byl Xamarin o mnoho efektivnější než hybridní aplikace, v matematických výpočtech a při čtecích operací dokonce téměř identický výkon, jako v případě nativní aplikace.

Kapitola 3

Xamarin

Xamarin je množina produktů, které se snaží pokrýt funkcionalitu majoritních mobilních operačních systémů a zpřístupnit dostupná jednotlivá API. Hlavními dvěma frameworky jsou **Xamarin.iOS** a **Xamarin.Android**, které umožňují rozběhnout aplikace psané v .NET technologii na telefonech s operačním systémem iOS respektive Android. Jsou postaveny na technologii Mono, což je open-source verze .NET Frameworku.

Oba frameworky jsou postaveny na podmnožině .NET Base class libraries (Standardních knihoven skupiny jazyků .NET) tzv. **Xamarin Mobile Profile**. Knihovny obsahují wrappery pro spouštění jednotlivých komponent SDK (Android, resp. iOS). Každá platforma poskytuje runtime, který pokrývá alokaci paměti, garbage collector, přilehlou interoperabilitu s runtime dané platformy apod.

3.1 Vývoj Xamarinu vedle .NET Framework

3.1.1 Mono

Počátkem Xamarinu je projekt Mono, který vznikl ve společnosti Xiami v roce 2001[12]. Byl navržený tak, aby splňoval standardy ECMA (Obecné standardy pro informační a komunikační systémy, zejména standardy struktury a fungování programovacích jazyků). Mezi tyto standardy patří také standard CLI (Common language infrastructure), což je společná infrastruktura programovacích jazyků. Hlavním inovátorem v této oblasti je společnost Microsoft, která aplikuje tyto standardy na své běhové prostředí (tzv. runtime) a popisuje vlastnosti proveditelného kódu spustitelného na daném runtime.

Toto běhové prostředí, tzv. CLR (Common language runtime) tvoří jádro .NET Framework, a cílem je docílit toho, aby na jedné platformě mohly fungovat různé zdrojové kódy bez nutnosti je přepisovat nebo přepisovat jejich překladače. CLI je specifikace, nikoliv implementace. Jedno z podmínek je, aby všechny jazyky kompatibilní s CLI byly kompilovány do CIL (Common intermediate language), nezávisle na hardwarové implementaci.

Součástí Mono je sada nástrojů kompatibilních právě s prostředím .NET, včetně kompilátoru jazyka C# a vlastní CLR, které umožňuje běh aplikací na počítačích s operačními systémy Unix, Windows i mac OSX. Součástí těchto nástrojů jsou i potřebné knihovny pro jednotlivé platformy (MonoTouch respektive Mono for Android), které vznikají v roce 2009. Díky open-source distribuci jsou postupně všechny nové funkcionality jazyka C# kontinuálně zapracovávány do těchto frameworků.

3.1.2 Xamarin framework

S rostoucím vývojem mobilních aplikací vzinkly ucelené frameworky Xamarin.Android v roce 2011 resp. Xamarin.iOS v roce 2013. „Naší vizí je umožnit vývojářům znovupoužít zdrojové kódy s jejich mechanismy a business logikou napříč všemi mobilními platformami a měnit pouze připojené uživatelské rozhraní pro platformně-specifické API.“[13]

V roce 2016 společnost kupuje Microsoft a zpřístupňuje zdrojové kódy pod MIT licencí. Xamarin spadá pod .NET Foundation (Nezávislá spolupracující organizace na podpoře .NET technologií), čímž se otevírá novým vývojářům. V dalším období je nejprve vývoj Xamarin integrován do stávajícího vývojového prostředí Visual Studio 2015 a následně již zcela vyvinut a vylepšován v prostředí Microsoft Visual Studio 2017 a Visual Studio for Mac.

3.2 Princip

3.2.1 Kompilace

Všechny Xamarin aplikace jsou při kompilaci překládány dvoufázově. První část kompilace je pro všechny platformy společná, a využívá standardu .NET platformy. Zdrojový kód v C# nebo F# jazyce je přeložen do tzv. MSIL(Microsoft Intermediate Language). Způsob jakým je tento jazyk následně překompilován do nativního kódu se liší v závislosti na platformě. Xamarin využívá dvou typů kompilování: Just in time a Ahead-of-time.

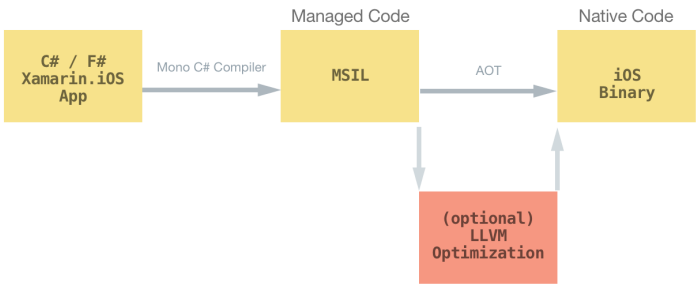
3.2.2 Just in Time kompilace (JIT)

Just in time kompilace přeloží MSIL kód do nativního binárního kódu při spuštění aplikace. Kompilace může být optimalizována pro cílový procesor a operační systém, v němž aplikace běží. JIT může vybrat takové instrukce, o nichž zjistí, že jsou daným procesorem podporovány. K zajištění tohoto stupně optimalizace statickým kompilátorem se musí buď přeložit binární kód pro každou zamýšlenou platformu či architekturu, nebo seskupit mnoho verzí částí kódu v rámci jednoho binárního kódu.

Systém JIT je schopen shromažďovat statistické údaje o tom, jak program aktuálně běží v prostředí, v němž je spuštěn, a může tak kód přeuspořádat a přeložit ho pro optimální výkon. Nicméně některé statické kompilátory mohou rovněž získat informace o cílovém běhovém prostředí.

3.2.3 Ahead of Time kompilace (AOT)

Operační systém iOS obsahuje omezení, které nedovoluje spouštět dynamicky generovaný kód za běhu. Z toho důvodu nelze kompilovat kód až při spuštění aplikace, ale ještě před tím. Toho je docíleno právě AOT kompilací, která se spustí při instalaci aplikace a vygeneruje nativní binární assembly (ARM). Před samotnou kompilací může ještě tzv. Linker optimalizovat některé odkazy v nativním kódu pro rychlejší přístup k jednotlivým částem, což má za následek například rychlejší načítání aplikace při otevření. Optimalizace viz. obrázek 3.1.



Obrázek 3.1: Schéma kompilace kódu typu Ahead of Time (AOT)
[14]

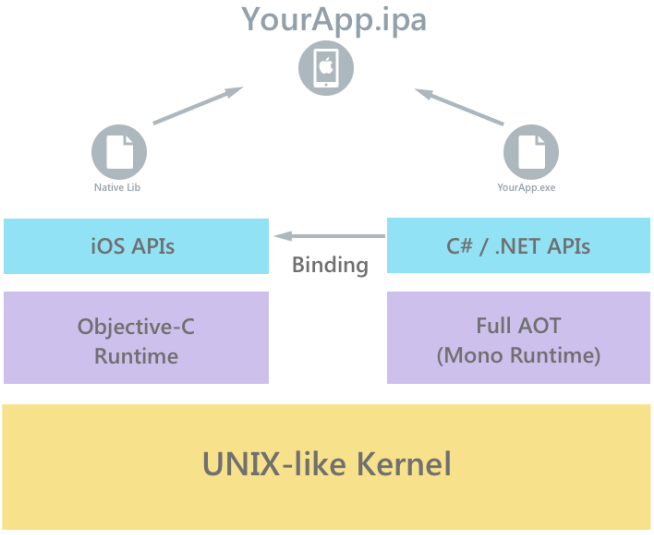
3.2.4 Xamarin.iOS

Aplikace napsaná pro Xamarin.iOS se skládá vždy z knihoven Mono, které umožňují běh C# kódu v prostředí operačního systému iOS a volání nativních funkcí, z knihoven, které naopak provádí nadstavbu o dostupné funkce v .NET Framework a vlastního aplikačního kódu. Způsob jakým je program zkompilován a spouštěn bude popsán v následujících odstavcích.

3.2.4.1 Schema aplikace

Xamarin.iOS aplikace běží na dvou běhových prostředích - Mono běhovém prostředí (Mono Runtime) a Objective-C Runtime. Používá AOT kompilaci ke kompilování C# kódu do nativního Objective-C. Obě běhová prostředí běží nad UNIX-ovým jádrem a vystavují dva typy API k využívání funkčnosti jak nativního[1] tak managed[2] systému.

[1]Kód, který běží přímo na procesoru daného zařízení
[2]Kód, který je spravovaný .NET Framework CLR - v případě Xamarin.iOS Mono Runtime



Obrázek 3.2: Schema Běhových prostředí Xamarin.iOS [15]

Dle schématu 3.2 je patrné, že musí existovat dvě varianty jak volat jednotlivé funkcionality, volání nativních funkcí operačního systému a rozšíření o funkce vlastní.

3.2.4.2 Selectory

Selektory jsou způsob jak volat vystavené nativní funkce v Objective-C pomocí jazyka C#. Ke komunikaci mezi běhovými prostředími se využívá tzv. binding, který umožňuje namapovat Xamarin na standardní iOS API. V Objective-C je toho docíleno pomocí **objc_msgsend funkcí**. Jedná se o speciální zprávy, které jsou posílány jednotlivým objektům. Xamarin.iOS tak obsahuje knihovny, které mapují veškeré API iOS systému.

3.2.4.3 Registrátory

Opačným směrem je potřeba spouštět komponenty, které v původním API nejsou. K tomu slouží tzv. registrátory, které zpřístupňují managed kód do Objective-C. Každá třída, která není součástí nativního rozhraní je zrcadlena jako nová Objective-C třída, s rozhraním, které používá třída původní (viz. ukázka kódu) a při kompilaci automaticky doplněny vygenerovaným kódem, který volá funkčnost napsanou v C#. Všechny třídy jsou potomky třídy NSObject v Objective-C, a při kompilaci jsou automaticky registrovány v binárním souboru.

3.2.4.4 Výstup aplikace

Binární soubor typu .app obsahuje spouštěcí sekvenci, která se vyvolá vždy při zavedení aplikace. Vstupním bodem je metoda xamarin_main, která inicializuje Mono runtime. Následně je v Mono prostředí spuštěna Main metoda Xamarin aplikace, která je jakousi obálkou pro nativní UIApplication.Main metodu. Main metoda je nabindována na UIApplication.Main pomocí registrátoru popsaného výše.

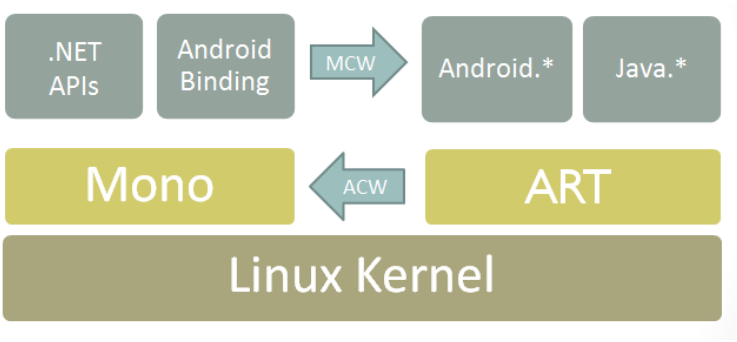
3.2.5 Xamarin.Android

Aplikace pro Xamarin.Android obsahuje Base class library, která je součástí Mono, jsou dostupné jak knihovny umožnující práci s .NET frameworkem tak různá API pro Android. V jádru Xamarin.Android je můstek, který umožňuje komunikaci mezi dvěma běhovými prostředími Virtuálním strojem Android Runtime (ART) a Mono Runtime. Běhové prostředí Mono je napsáno v jazyce C.

3.2.5.1 Schema aplikace

Základem aplikace jsou dvě běhová prostředí, která spolu navzájem komunikují. Většina systémových funkcí není přímo dostupná nativním aplikacím, ale jsou pouze vystavena jejich rozhraní v knihovnách Java.* a Android.*

Aplikace obsahuje knihovnu Mono.Android.dll, která pokrývá struktury pro svázané Android API a komunikaci s runtime virtuálním strojem.



Obrázek 3.3: Schema Běhových prostředí Xamarin.Android [16]

3.2.5.2 JNI

Technologie, používaná jako můstek pro komunikaci mezi rozhraními se nazývá Java native interface (JNI). JNI je rozhraní, které umožňuje propojit kód běžící ve virtuálním javovském stroji(JVM) s jinými programy, které nejsou napsané v Javě. Bytecode zkompilovaný do Javy obsahuje hlavičky a signatury volatelných objektů, které se dají volat po načtení nativní knihovny.

3.2.5.3 Android volatelné wrappery

Android volatelné wrapery (Android callable wrappers) jsou můstky, které používá Android runtime v případě potřeby volat metody v managed kódu (V případě Xamarinu napsané v C#, spravované Mono runtime). Managed kód implementuje požadované rozhraní (stejná base class, seznam implementovaných rozhraní, konstruktor a implementace předepsaných metod) a pomocí JNI technologie jsou tyto metody vzdáleně volány z JVM vytvořenými wrappery. Wrappery jsou vytvořeny během kompilace procesem monodroid.exe a přiloženy do byte kódu aplikace.

3.2.5.4 Managed volatelné wrappery

Managed wrappery jsou JNI můstkem, který se používá pro vyvolání Android kódu v JVM, nebo v případě implementace existujících rozhraní.Wrapper je zodpovědný za vzájemnou převoditelnost Android typů a nově vytvořených v Xamarinu a volání podružené Android vrstvy v Managed kódu.Každý wrapper v managed kódu si drží globální referenci na svůj Java protějšek.

3.2.5.5 Výstup aplikace

Výstupem aplikace Xamarin.Android je speciální ZIP balíček s koncovkou .apk, který obsahuje stejnou strukturu jako běžné Android aplikace napsané v Javě s několika rozšířeními. První z nich jsou jednotlivé nezkomprimované assembly aplikace - sem patří samotná aplikační logika, popřípadě knihovny třetích stran napsané pro Xamarin (V jazyce C# nebo F#) ve složce assemblies. Při spuštění aplikace jsou tyto assembly nahrané do procesu z paměti a spouštěné v Mono runtime.

Nepřítomnost komprimace má 2 důsledky, zaprvé se aplikace při načítání rychleji spustí, neboť není potřeba zpětná dekomprese, zadruhé jsou tyto balíčky větší oproti nativním aplikacím. Dále balíček obsahuje knihovny pro samotný Mono runtime v závislosti na architektuře hardware zařízení.

3.3 Struktura projektu v Xamarin

Xamarin je rozsáhlý Framework, který umožňuje vytvářet aplikace několika způsoby. Všechny mají společný vývoj v jazycích C#, potažmo XAML, nicméně vývojář má možnost si zvolit přístup pro vývoj společné logiky i celého grafického uživatelského rozhraní. V této kapitole bude popsán vývoj aplikace v Xamarinu všemi způsoby, a celý vzniknuvší projekt je součástí příložého disku (projekt **./TestApp**).

Projekt celé aplikace v .NET se nazývá **Solution**[3], a skládá se z jednoho až několika dalších dílčích projektů. Filosofií vývoje všech aplikací (nejen mobilních) v jazycích skupiny .NET je vytvářet co nejmenší projekty, které budou implementovat jednu primární funkcionalitu. Knihovní projekty by měly být vzájemně nezávislé a znovupoužitelné, podle obecných pravidel vývoje DRY[4].

Dílčí projekty mohou být do Solution připsány přímo vývojářem aplikace (V takovém případě má vývojář nad projektem absolutní kontrolu, může ho modifikovat a upravovat, a zároveň může kompletní strukturu zkoumat při debuggování) nebo přiloženy jako zásuvné moduly – tzv. **Reference**. Tyto moduly jsou poskytovány třetími stranami a mohou pokrývat podpůrné funkčnosti, které slouží k dosažení funkcionality výsledné aplikace. Tyto moduly, tzv. assemblies, jsou již zkompilované projekty, na které lze z projektu odkazovat.

3.3.1 Xamarin.iOS, Xamarin.Android

Základním stavebním kamenem aplikace Xamarin je specifický projekt, obsahující jednotlivé soubory pro konkrétní platformu. Vyvíjet lze v každém projektu odděleně. Jediný společný prvek v takovém případě bude jazyk C#. Struktura jednotlivých projektů se naprosto liší – odpovídá struktuře aplikace na cílové platformě s rozšiřujícími soubory Xamarinu. Takový projekt je poté kompilátorem přeložen do **.apk** balíčku v případě Androidu, respektive do **.ipa** balíčku v případě iOS, včetně všech jeho závislostí, stejně jako v případě nativního vývoje. Tento balíček je následně používán při instalacích do fyzických zařízení.

3.3.2 Struktura projektu Xamarin.Android

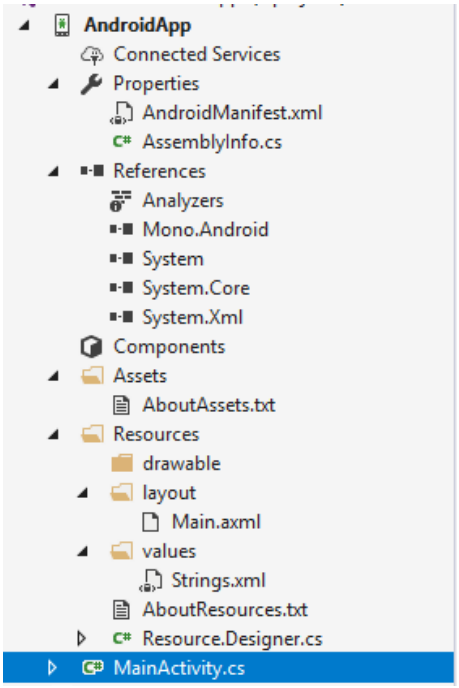
Základní projekt Xamarin.Android obsahuje následující komponenty:

- **Properties** – složka obsahující základní konfiguraci projektu (Název projektu, číslo verze apod.) a ***AndroidManifest.xml***, který je konfiguračním souborem pro Android a udává základní vlastnosti výsledné aplikace (minimální verze OS mobilního zařízení, požadovaná oprávnění, název aplikace, ikonu atd.)
- **References** – dle základní struktury každé C# aplikace sem patří všechny knihovny a projekty, na kterých je cílový projekt závislý. V případě větších projektů tento způsob umožňuje odkazovat více projektům na jednu knihovnu bez nutnosti jejího duplikování. Základními referenčními knihovnami pro projekt Android aplikace jsou
  - **Mono.Android** – Knihovna obsahující veškeré API Android, zabalené do C# , stejně jako samotné běhové Mono prostředí pro spouštění nově vytvořeného kódu a mechanismy pro spolupráci mezi běhovými prostředími.
  - **System, System.Core, System.Xml** – Základní knihovny pro zavedení aplikace a práci se stavebními soubory aplikace (AndroidManifest, zdroje atd.)
- **Assets** – složka pro vkládání externích příloh do aplikace (textové soubory, hudební soubory). Soubory jsou ve výsledném balíčku přiloženy přímo k aplikaci.
- **Resources** – složka pro vkládání zdrojů pro aplikaci. Mezi zdroje lze zařadit:
  - **drawable** – libovolná ikona nebo obrázek, který lze zařadit do složky rozdělené podle *výsledné velikosti* ikony na zařízení (Na tabletu a malém mobilu tak může být různě kvalitní ikona) nebo podle *lokalizace* (Dle předvoleného jazyka operačního systému na mobilním zařízení lze zobrazit jinou ikonu – například vlajku)

[3]Překlad pro toto slovo je "Řešení", ale většinou se nepřekládá, jelikož jde o zaběhlý pojem
[4]DRY - Do not repeat yourself, pravidlo vývoje aplikací, aby se kód se stejnou dunkcionalitou napsal jen jednou

  - **layout** – layout je základní stavební prvek grafického rozhraní, který udává, jak budou jednotlivé ovládací prvky rozloženy na obrazovce. Složka layout může obsahovat rozložení pro různé obrazovky. Tyto soubory s koncovkou **.axml** jsou definovány pro Android.
  - **values** – v XML souborech v této složce lze definovat konstantní hodnoty, které se využívají jak v android manifestu, tak v layoutu. Jejich účel je v zásadě dvojí:
    * zamezit duplicitám
    * umožnit lokalizaci stejně jako v případě ikon na základě předvoleného jazyka
    * definovat styly (barvy, velikosti) ovládacích prvků a barevné téma

- **MainActivity.cs** – třída, která je vstupním bodem do aplikace Xamarin.Android. Název MainActivity není vyžadován, nicméně minimálně jedna třída, která je potomkem třídy ***Android.App.Activity*** musí být v projektu zahrnutá.



Obrázek 3.4: Základní struktura projektu Android

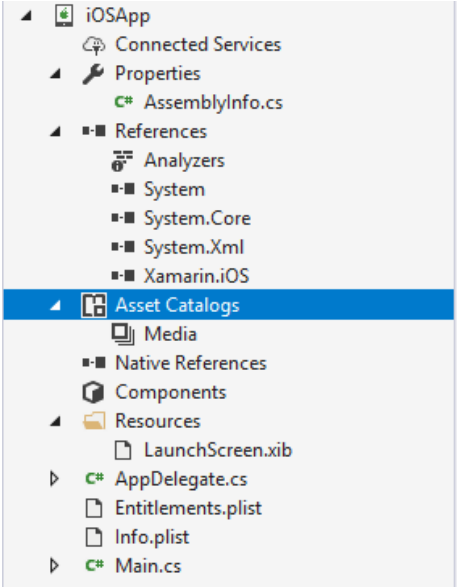
3.3.3 Struktura projektu Xamarin.iOS

Základní projekt iOS aplikace má následující strukturu:

- **Properties** – složka obsahující základní konfiguraci projektu (Název projektu, číslo verze apod.)
- **References** – dle základní struktury každé C# aplikace sem patří všechny knihovny a projekty, na kterých je cílový projekt závislý. V případě větších projektů tento způsob umožňuje odkazovat více projektům na jednu knihovnu bez nutnosti jejího duplikování. Základními knihovnami pro projekt iOS aplikace jsou
  - **Xamarin.iOS**– knihovna obsahující veškeré API iOS, zabalené do C# , stejně jako samotné běhové Mono prostředí pro spouštění nově vytvořeného kódu a mechanismy pro spolupráci mezi běhovými prostředími.
  - **System, System.Core, System.Xml** – Základní knihovny pro zavedení aplikace a práci se stavebními soubory aplikace
- **Asset Catalogs** - složka pro vkládání zdrojů pro aplikaci. Mezi zdroje lze zařadit:

  - **AppIcons** – Ikony na pracovní plochu mobilního zařízení v různých velikostech v závislosti na zařízení, na které je aplikace instalována
  - **LaunchImages** – Načítací obrazovka pro aplikaci
  - **Media**
    * Image – Obrázek nebo ikona. Každou ikonu lze přidat ve 4 různých velikostech (Vektorově, 1x, 2x a 3x)
    * Color – statická hodnota barvy
    * Data – extra přiložený soubor

- **Info.plist** – Speciální XML konfigurační soubor, který definuje základní informace o aplikaci (Název, verze, minimální verze OS apod.), požadovaná oprávnění, vlastní datové typy apod.
- **Entitlemens.plist** – Speciální konfigurační soubor, který se využívá k nastavení využívání speciálních funkcionalit Apple zařízení (Wallet, HomeKit, Siri, Push Notifikace atd.)
- **Main.cs** – Vstupní bod aplikace, musí obsahovat metodu main, ze které se volá instance aplikačního delegáta (třída, která zajišťuje běh aplikace)
- **AppDelegate.cs** – Třída, která je potomkem třídy **UiKit.UIApplicationDelegate**, a jedná se o vstupní bod aplikace pro iOS.



Obrázek 3.5: Základní struktura projektu iOS

3.3.4 Sdílené projekty

3.3.4.1 Shared projekt

První variantou pro sdílení shodného kódu, která se využívá pro aplikace na obě zmiňované platformy, je tzv. **Shared** projekt**.** Tento typ není plnohodnotným projektem, ale pouze obálkou pro různé soubory z .NET technologie (souborový typ .cs, .resx, .xaml apod.). Při kompilaci se soubory tváří jako součást projektů, které na něj přímo odkazují ve svých referencích. Tudíž veškeré závislosti, které jsou v souborech v Shared projektu použity musí být zahrnuty v referencích v kompilovaném projektu. Z tohoto důvodu projekt neobsahuje žádné další potřebné speciální soubory, ale pouze ty, které do něj přibydou v době vývoje.

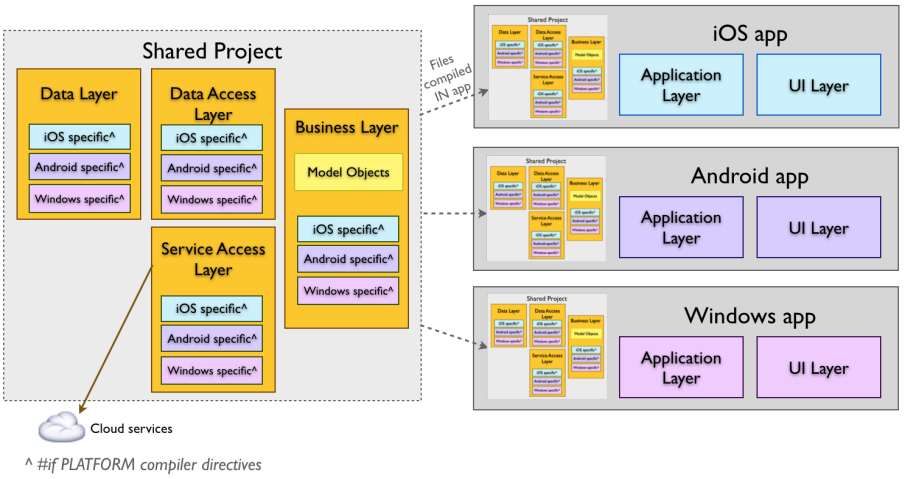
Pakliže je potřeba v některé konkrétní třídě oddělit specifickou implementaci pro iOS resp. Android aplikaci, může se ve třídě využít direktiv kompilátoru (viz. zdrojový kód direktiv), které zajistí, že v době kompilace bude využita jen část kódu určena pro cílovou platformu. Sekce direktivy jsou uvedeny klíčovou sekvencí: #if $PLATFORMA, #endif, kde $PLATFORMA je nahrazena názvem platformy pro kterou má být sekce kompilována. Konvencí pro projekty Android a iOS

jsou přednastavené proměnné ***__ANDROID__*** resp. ***__IOS__*** ve vlastnostech cílového projektu, ale je možné využít libovolných proměnných pro různé projekty. Direktiv kompilátoru lze využít i v dalších typech sdílených projektů, nejčastější užití je ale právě v projektu typu Shared.

Ukázka využití direktiv kompilátoru v Shared projektu:

```
namespace TestApp
{
    public class DirectiveClass
        {
        public string GetPlatformSpecificImplementationmethod()
        {
            string platformName = null;

#if __IOS__
            platformName = "IOS";
#endif

#if __ANDROID__
            platformName = "ANDROID";
#endif
            return platformName;
        }
    }
}
```

Jak již bylo zmíněno, Shared projekt se využívá pro sdílení stejného zdrojového kódu, zejména společné logiky, případně volání webových služeb, práce s databází apod. Nehodí se pro práci s grafickým rozhraním, neboť implementace jednotlivých ovládacích prvků je na každé platformě odlišná.



Obrázek 3.6: Architektura aplikace se sdíleným kódem v Shared projektu [17]

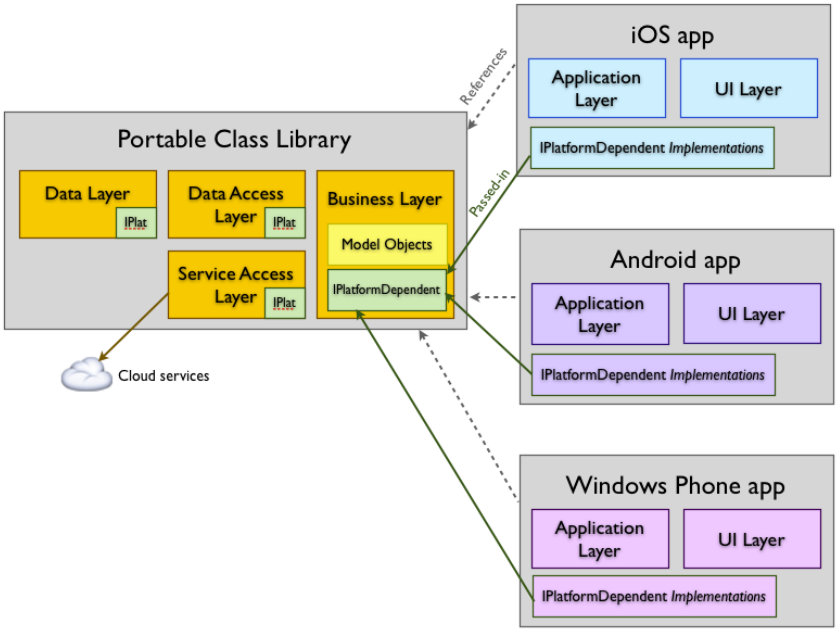
3.3.4.2 PCL a .NET Standard

Vedle platformních projektů pro jednotlivé operační systémy pak Solution může obsahovat i projekty se sdíleným kódem, který je při běhu spouštěn právě v Mono runtime. Sdílený projekt obsahuje navíc referenci na .NET SDK, což je soubor standardních knihoven .NET. Pro vývoj aplikace v Xamarin se využívají dva typy sdílených projektů - PCL a .NET Standard.

3.3.4.3 PCL

Portable class libraries neboli PCL jsou projekty obsahující soubor knihoven s určitou podmnožinou ze standardní funkčnosti jazyka C#, tzv. Base class libraries (BCL). Při kompilaci probíhá překlad těchto knihoven pro konkrétní cílovou platformu, neboť implementace se na různých operačních

systémech může lišit. Každá platforma implementuje svůj tzv. profil, tj. podmnožinu tříd, které lze na dané platformě využít. Sdílený projekt PCL lze využít napříč platformami, kde každá pokrývá určitý profil. Rozdílné platformy, implementující stejný profil mohou využívat stejný PCL projekt definující společné rozhraní.

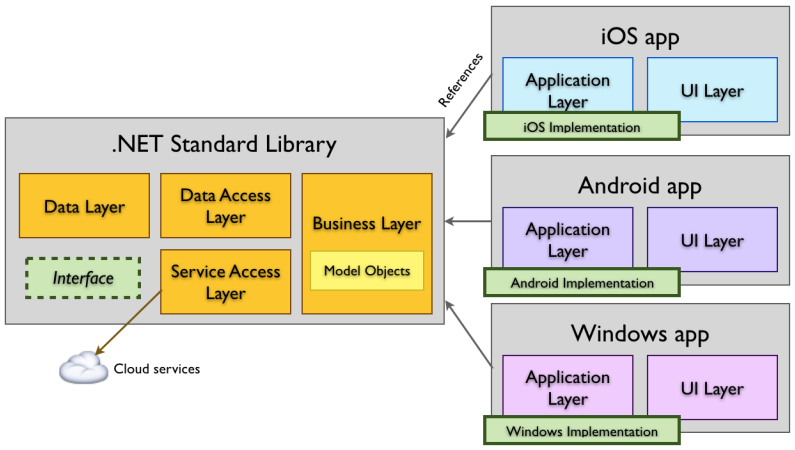


Obrázek 3.7: Architektura aplikace se sdíleným kódem v PCL projektu [18]

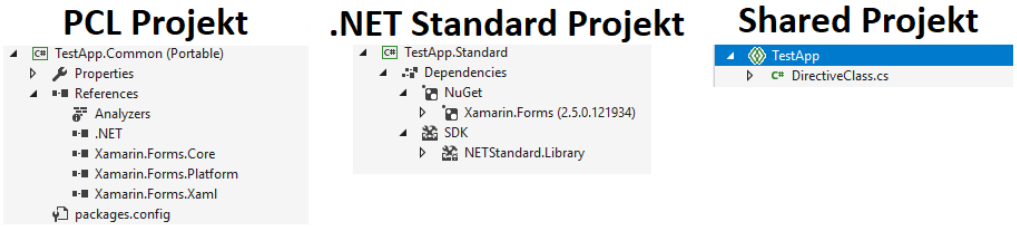
3.3.4.4 .NET Standard

V roce 2016 vydal Microsoft nový návrhový vzor pro sdílení kódu napříč platformami tzv. .NET Standard, který má PCL zcela nahradit, implementace tohoto návrhového vzoru pro Xamarin byla vydána v létě roku 2017. Oproti PCL se .NET Standard liší hlavně v pokrytí funkcionalit .NET. Vývoj samotné aplikace eRecept byl zpočátku realizován pomocí PCL projektů, které byly následně převedeny do .NET Standard technologie.

.NET Standard je formální specifikace[19] .NET API, které bude dostupné na všech .NET běhových prostředích. To znamená, že knihovna napsaná v této technologii může být využita v projektech libovolné platformy (Xamarin, UWP, Silverlight, .NET Core atd.), což dramaticky zvýší znovu použitelnost jednotlivých knihoven a možnost využít již existujících knihoven třetích stran ve svých aplikacích.



Obrázek 3.8: Architektura aplikace se sdíleným kódem v .NET Standard projektu [19]



Obrázek 3.9: Porovnání struktury sdílených projektů s odkazem na Xamarin.Forms knihovnu (v Shared projektu nejsou přímé odkazy a reference)

3.4 Xamarin.Forms

Framework, který umožňuje psát sdílený kód i pro grafické ovládací prvky, nebo například sjednotit implementaci speciálních funkcionalit na rozdílných platformách do jednoho projektu se nazývá **Xamarin.Forms**. Jde o soubor tříd a dalších datových struktur, který implementuje obecné koncepty mobilních aplikací na jednotlivých platformách a vystavuje nad nimi rozhraní pro jednotné použití.

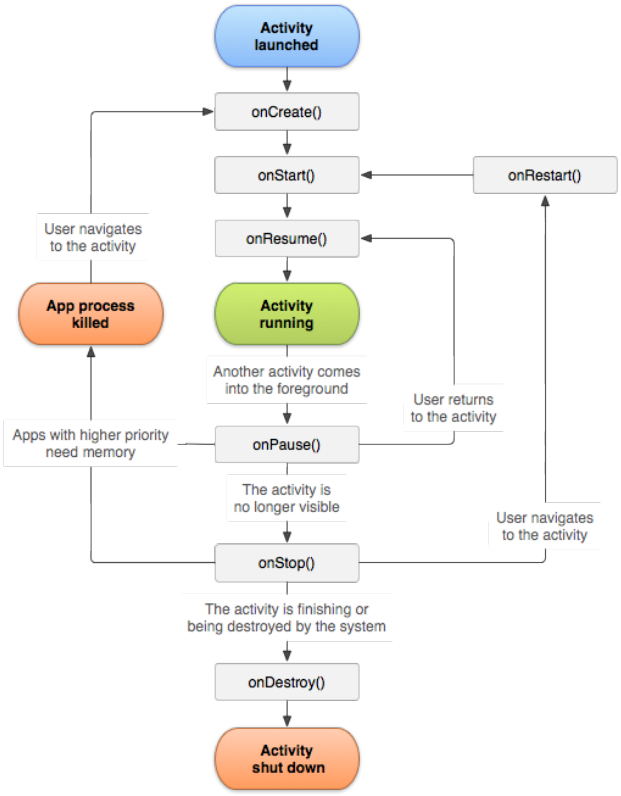
Pokud chceme využívat funkcionalit frameworku, pak je potřeba, aby všechny projekty (jak platformní tak sdílený) měly ve svých referencích Xamarin.Forms knihovny, které zabalují rozhraní a implementaci společných konceptů mobilních aplikací popsané v následujících podkapitolách.

3.4.1 Životní cyklus aplikace

Aplikace má svůj životní cyklus v rámci svého běhu na cílovém mobilním zařízení. Obě platformy mají odlišný způsob práce s aplikací, takže jejich porovnání nelze udělat naprosto implicitně. Obě platformy rozlišují v zásadě 4 stavy aplikace:

- **Neběžící** – Aplikace nebyla zavedena do paměti, a nebyla ani spuštěna (výchozí bod)
- **Spuštěna na popředí** – Aplikace běží na popředí operačního systému, lze s ní pracovat a využívat jejích funkcionalit
- **Spuštěna na pozadí** – Aplikace běží na pozadí. V tomto případě mohou nastat dva případy, a to, pokud aplikace provádí nějaký dílčí úkol (po jeho dokončení může například upozornit na potřebu uživatelské interakce) anebo je čistě uspána na úkor jiné aplikace.
- **Ukončena** – Aplikace je ukončena, tedy už není v paměti. V takovém případě jsou všechny využívané prostředky (paměť, hardware, vlákna procesoru) uvolněny zpět pro operační systém

Mezi vyjmenovanými stavy může aplikace libovolně přecházet, a na jednotlivé změny lze v jejím průběhu reagovat. Porovnání přechodů mezi stavy na obou platformách je patrné na obrázcích 3.10 a 3.11.



Obrázek 3.10: Životní cyklus aktivity (základní stavební blok aplikace) v Android [20]

3.4.1.1 Události Android

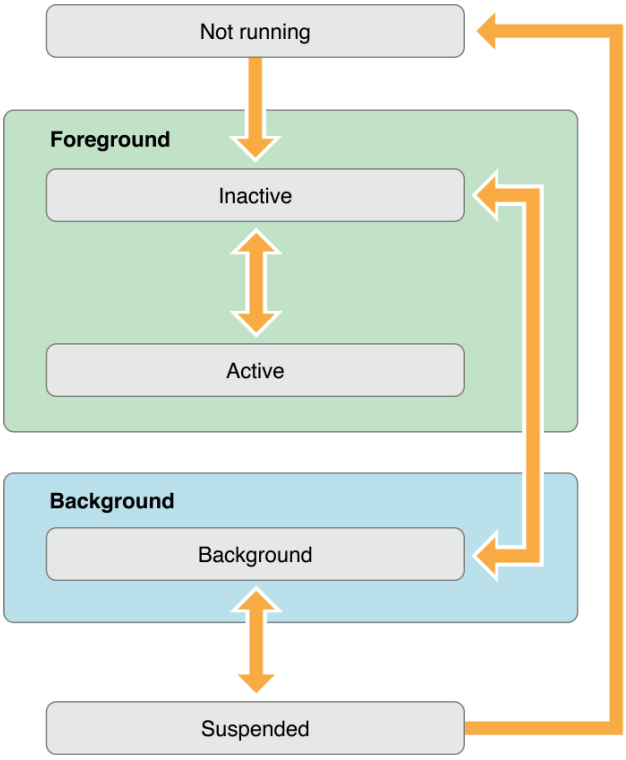
Události Aktivity, na které může reagovat, jsou:

- **onCreate** – **Z**avolá se v momentě kdy je aktivita poprvé vytvořena
- **onStart** – Zavolá se v momentě zobrazení aktivity na popředí
- **onResume** – Zavolá, se pokud aplikace vyžaduje interakci uživatele
- **onPause** – Zavolá se, pokud je aplikace skrytá (jinou aplikací nebo interakcí OS)
- **onStop** – Zavolá se, pokud je aplikace pozastavena (není dále viditelná)
- **onRestart** – Volá se, pokud byla aplikace zastavena a má být znovu spuštěna
- **onDestroy** – Volá se před tím, než je aplikace odstraněna z paměti

3.4.1.2 Události iOS

Události, na které může aplikace reagovat, jsou:

- **application:willFinishLaunchingWithOptions:** — Zavolá se v době startu aplikace
- **application:didFinishLaunchingWithOptions:** — Zavolá se těsně předtím, než je aplikace po spuštění zobrazena na displej. Hodí se pro finální inicializace
- **applicationDidBecomeActive:** — Zavolá se předtím, než je aplikace zobrazena – pokaždé pokud je volána z pozadí



Obrázek 3.11: Životní cyklus aplikace v iOS
[21]

- **applicationWillResignActive:** — Volá se v případě opuštění aktivního stavu na popředí
- **applicationDidEnterBackground:** — Událost oznamující, že aplikace nyní běží na pozadí
- **applicationWillEnterForeground:** — Aplikace je připravena přejít z pozadí do popředí
- **applicationWillTerminate:** — Aplikace byla ukončena (operačním systémem nebo uživatelem)

3.4.2 Vstupní bod

Nad oběma operačními systémy staví obecnou strukturu framework Xamarin.Forms prostřednictvím třídy **Application**, která je vstupním bodem v případě sdíleného kódu. Třída se instancuje v konkrétním platformním projektu Android nebo iOS. Tato třída je většinou ve sdíleném projektu, a umožňuje základní funkcionalitu aplikace. Do této třídy se zapisuje i úvodní obrazovka po spuštění aplikace, mohou zde být uplatněné obecné styly ovládacích prvků a chování aplikace během jejího životního cyklu:

- **OnStart** – Volá se v momentě, kdy je aplikace spuštěná.
- **OnSleep** – Volá se pokaždé, když je aplikace skryta na pozadí.
- **OnResume** – Volá se ve chvíli, kdy aplikace přechází z pozadí do popředí.

Výchozí aplikace, která bude využívat sdílený kód v projektu typu .NET Standard, a bude zobrazovat pouze jednoduchý text vypadá následujícím způsobem (zobrazeny jsou pouze relevantní zdrojové kódy a soubory, pro celou implementaci lze prohlédnout projekt Concepts.sln na přiloženém disku **./Concepts**):

.NET Standard projekt (Concepts)
App.cs (./)

```
using Xamarin.Forms;

namespace Concepts
{
    public class App : Application
    {
        public App()
        {
            MainPage = new ContentPage()
            {
                Padding = Device.OnPlatform(new Thickness(20), new
                    Thickness(), new Thickness()),
                Content = new Label()
                {
                    Text = "Hello world",
                }
            };
        }
    }
}
```

V konstruktoru třídy musí být nastavena property **MainPage**, která udává výchozí obrazovku celé aplikace, podrobnější popis jednotlivých ovládacích prvků v kapitole 3.4.5. **Android projekt (Concepts.Android)**
styles.xml (./resources/values/)

```
<?xml version="1.0" encoding="utf-8" ?>
<resources>
  <style name="MainTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <item name="windowNoTitle">true</item>
  </style>
</resources>
```

Projekt Android musí obsahovat soubor styles.xml, který určuje základní vlastnosti stylů ovládacích prvků a barevné téma. V tomto příkladě je zobrazeno naprosté minimum nutné ke správnému spuštění aplikace.
MainActivity.cs (./)

```
using Android.App;
using Android.OS;
using Concepts;
using Xamarin.Forms.Platform.Android;

namespace Concepts.Android
{
    [Activity(Label = "Concepts", Theme = "@style/MainTheme",
        MainLauncher = true)]
    public class MainActivity : FormsAppCompatActivity
    {
        protected override void OnCreate(Bundle savedInstanceState)
        {
            base.OnCreate(savedInstanceState);
            Xamarin.Forms.Forms.Init(this, savedInstanceState);
            LoadApplication(new App());
        }
    }
}
```

Třída MainActivity je vstupním bodem celé aplikace, konkrétně v metodě **OnCreate** je potřeba inicializovat instanci sdílené aplikace. Třída má tzv. atribut Activity, v tom je potřeba nastavit následující 3 vlastnosti:

- **Label** – Zobrazovaný popisek u ikonky na ploše mobilního zařízení

- **Theme** – Definice základního tématu celé aplikace (odkazuje na soubor styles.xml)
- **MainLauncher** – Příznak, že tato aktivita je výchozí aktivitou aplikace. Dle koncept Android aplikací může mít aplikace více aktivit, pro aplikace Xamarin.Forms lze všechno řešit v jedné aktivitě.

Ještě před inicializací instance Application třídy je potřeba inicializovat knihovny Xamarin.Forms (zjednodušeně řečeno se datové struktury Xamarinu zavedou do paměti, takže mohou být následně instancovány pomocí JNI – viz. kapitola 3.2.5.2.

iOS (Concepts.iOS)
AppDelegate.cs (./)

```
using Foundation;
using UIKit;
using Xamarin.Forms.Platform.iOS;

namespace Concepts.iOS
{
    [Register("AppDelegate")]
    public class AppDelegate : FormsApplicationDelegate
    {
        public override bool FinishedLaunching(UIApplication
            application, NSDictionary launchOptions)
        {
            Xamarin.Forms.Forms.Init();
            LoadApplication(new App());
            return base.FinishedLaunching(application, launchOptions);
        }
    }
}
```

Třída AppDelegate je vstupním bodem aplikace pro iOS. Atributem nad touto třídou se registruje třída k překladu viz. kapitola. Obdobně jako u Android aplikace, je potřeba inicializovat knihovny Xamarin.Forms a následně inicializovat instanci třídy aplikace.

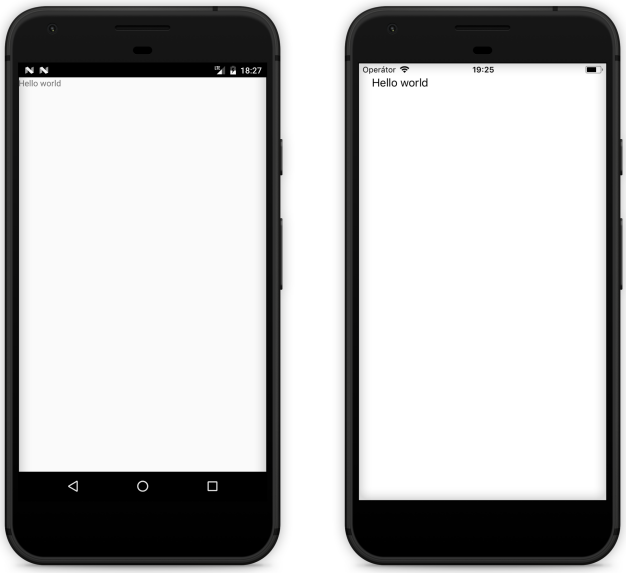
Info.plist (./)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
    "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>CFBundleDisplayName</key>
        <string>Concepts</string>
```

V souboru Info.plist jsou další konfigurační informace týkající se kompatibility s verzemi operačního systému, oprávnění a další. V tomto případě je jen důležité zmínit, že se zde nastavuje i textový popisek ikonky na ploše mobilního zařízení pomocí páru klíč-hodnota **CFBundleDisplayName**.

3.4.3 MVVM

MVVM neboli Model-View-ViewModel je návrhový vzor pro grafické aplikace, který rozděluje datovou strukturu, aktuální stav od uživatelského rozhraní ve třech úrovních. Hlavní myšlenka MVVM je prostá – vytvořit třídu, která si drží stav aplikace. Nazývá se ViewModel. Té se dotazuje uživatelské rozhraní, které podle ní vykresluje ovládací prvky. A naopak zadá-li uživatel do uživatelského rozhraní nějaké údaje, zpropagují se automaticky do ViewModelu[22]. ViewModel je nejdůležitější třída. Poskytuje všechna data pro uživatelské rozhraní, které se nazývá View. Důležité je, že poskytuje svá data v takových datových strukturách, které vyvolávají události při jejich změně. To umožňuje uživatelskému rozhraní nová data automaticky zobrazit hned jak se ve ViewModelu změní. ViewModel má dva základní kameny. Prvním je kolekce ObservableCollection<T>, která hlásí, když je přidán nebo odebrán její prvek. Druhým je rozhraní INotifyPropertyChanged. Implementace takového rozhraní musí obsahovat událost, která nastane, když se změní některá z vlastností ViewModelu.



(a) (b)

Obrázek 3.12: Výchozí obrazovka aplikace Concepts na Android (a) a iOS (b)

3.4.3.1 Model

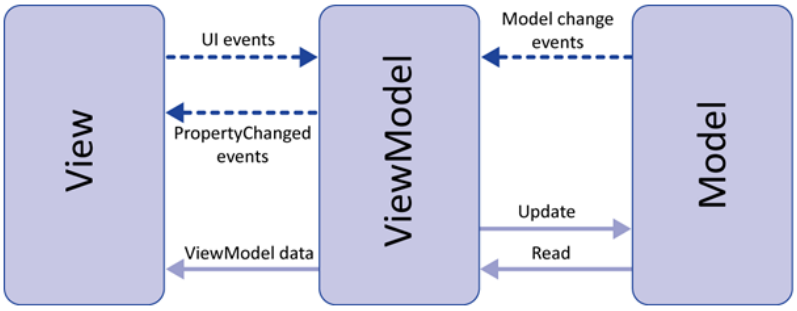
Popisuje data, se kterými aplikace pracuje. Jedná se o třídy entit, která nějakým způsobem popisují data. Většinou jsou entity svou strukturou shodné s tabulkami přidružené databázi, pro jejich snažší uchovatelnost. Tyto entity nazáváme Modely. Model nesmí o stavu ovládacích prvků nic vědět.

3.4.3.2 View

Reprezentuje uživatelské rozhraní (v jazyce XAML nebo v C#). Může se jednat o okno aplikace, stránku, nebo ovládací prvek. Označuje se také jako UI, formulář, nebo prezentační vrstva, ale jde pořád o to samé.

3.4.3.3 ViewModel

Spojuje Model a View a drží si stav aplikace. Ovládací prvky jsou pomocí bindingu (viz. 3.4.4) propojeny s ViewModelem a čerpají z něj svůj obsah. Provádí se v něm filtrování dat v závislosti na stavu aplikace. Aby se jeho vlastnosti propagovaly do View, musí implementovat rozhraní INotifyPropertyChanged. Jeho vlastnosti typu kolekce musí obdobně implementovat rozhraní *INotifyCollectionChanged*.



Obrázek 3.13: Vzájemná vazba jednotlivých vrstev v MVVM [23]

3.4.3.4 XAML

XAML je modifikace XML jazyka pro .NET technologie pro instancování a inicializaci grafických objektů ve strukturované formě. XAML je dobrým nástrojem pro implementaci View vrstvy v návrhovém vzoru MVVM. Propojení s ViewModel vrstvou se provádí pomocí tzv. DataBindingu. Ačkoli to návrhový vzor přímo nedoporučuje, určitou logiku práce View lze psát v tzv. Code-behind dané třídy. Soubory s koncovkami .xaml a .xaml.cs tvoří společně nedělitelný pár, který v době kompilace vytvoří datový typ, tak jakoby celá třída byla napsána pouze v běžném kódu.

Výhody

- XAML je čitelnější a intuitivnější pro návrh grafického rozhraní než kód
- Struktura předek-potomci je díky XML více zjevná než v případě kódu
- Grafické prvky mohou být psané jak vývojáři, tak pomocí různých grafických nástrojů

Nevýhody

- XAML nemůže obsahovat žádný kód – veškerý kód se děje v dílčím souboru code-behind
- Neobsahuje mechanismy pro smyčky (Pro zobrazení dat z kolekce lze ale využít mechanismy, které zobrazí data se stejnou vizuální úpravou – např. ItemsSource vlastnost u ovládacího prvku ListView)
- XAML nemůže volat metody – definuje pouze odkazy na události, které jsou reakcemi na uživatelskou interakci
- XAML nemůže instanciovat třídy bez neparametrického konstruktoru (všechny zamýšlené ovládací prvky tedy musí mít alespoň prázdný konstruktor bez aprametrů)

.Portable projekt (EreceptSamples)
XamlPage.xaml. (./Xaml/)

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="EreceptSamples.Xaml.XamlPage"
             Title="XAML"
             Padding="20"
             BackgroundColor="LightGreen">
    <StackLayout BackgroundColor="LightGoldenrodYellow" Padding="5">
        <StackLayout Orientation="Horizontal"
            VerticalOptions="FillAndExpand"
            HorizontalOptions="FillAndExpand">
            <BoxView VerticalOptions="FillAndExpand"
                HorizontalOptions="FillAndExpand" Color="Black"/>
            <ContentView BackgroundColor="Aqua"
                VerticalOptions="FillAndExpand"
                HorizontalOptions="FillAndExpand" />
        </StackLayout>
        <StackLayout>
            <ListView x:Name="MyListView">
                <ListView.ItemTemplate>
                    <DataTemplate>
                        <TextCell Text="{Binding .}"/>
                    </DataTemplate>
                </ListView.ItemTemplate>
            </ListView>
            <Button Text="Tlačítko" VerticalOptions="EndAndExpand"
                HorizontalOptions="CenterAndExpand"/>
        </StackLayout>
    </StackLayout>
</ContentPage>
```

Příklad zdrojového kódu v XAML.

3.4.4 Data binding

Data binding je mechanismus, který umožňuje vytvořit pouto mezi View a ViewModelem. S daty lze manipulovat přímo na ViewModelu. O překreslení a aktualizaci dat se postará právě Data binding, stejně opačným směrem lze ve View pracovat s ovládacími prvky, a data se promítnou skrze binding do ViewModelu. DataBinding je tedy vždy mezi dvojicí tzv. **Source** a **Target**(Zdroj a Cíl), tzv. BindableProperty a konkrétní Property na ViewModelu, a lze mezi nimi vytvořit jednosměrnou nebo obousměrnou vazbu. Například u textového popisku je žádoucí, aby text byl závislý na hodnotě ve ViewModelu, naopak u vstupního pole chceme aby se hodnota zapsaná uživatelem promítla i do ViewModelu. Proto existují 4 typy směřování Data bindingu:

- **Default** – Základní hodnota, nastavená v Bindable property
- **OneWay** – Data jsou směřována ze Zdroje k Cíli
- **OneWayToSource** – Data jsou směřována od Cíle ke zdroji
- **TwoWay**– Data jsou směřována obousměrně (změna se projeví jak ve ViewModelu tak ve View)

Následující příklad demonstruje jednoduché vytvoření vazby mezi ovládacími prvky a modelem:

.NET Standard projekt (Concepts)
BindingModel.cs (./DataBinding/)

```
namespace Concepts.DataBinding
{
    public class BindingModel
    {
        public string Nazev { get; set; }
        public string Poznamka { get; set; }
    }
}
```

Jednoduchý model o dvou vlastnostech Název a Poznámka.

BindingViewModel.xaml (./DataBinding/)

```
namespace Concepts.DataBinding
{
    public class BindingViewModel : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;

        public BindingModel Item { get; set; }

        private int timer;
        public int Timer
        {
            get
            {
                return timer;
            }
            set
            {
                timer = value;
                OnPropertyChanged();
            }
        }
        public BindingViewModel()
        {
            Item = new BindingModel() { Nazev = "Položka – Ibalgin" };
            Timer = 0;
            Device.StartTimer(new System.TimeSpan(0, 0, 1), () => {
                Timer += 1; return true; });
        }
```

```
        private void OnPropertyChanged([CallerMemberName]string
            callerName = null)
        {
            PropertyChanged?.Invoke(this, new
                PropertyChangedEventArgs(callerName));
        }
    }
}
```

ViewModel v tomto příkladě je potomek třídy **INotifyPropertyChanged,** která je zodpovědná za notifikaci v případě změny hodnoty na ViewModelu nad danou vlastností. ViewModel má vlastnost **Item**, což je typ BindingModel pro ilustraci přenosu dat mezi View a Modelem skrze ViewModel a dále **Timer** typu integer, pro ilustraci průběžné aktualizace na základě hodnoty. Rozhraní definuje událost **PropertyChanged**, která se vyvolá při změně vybrané vlastnosti, v tomto případě právě Timeru. Při vyvolání této události je View upozorněno na změnu a dojde k překreslení. Metoda **Device.StartTimer** vytvoří nové vlákno, které každou vteřinu aktualizuje hodnotu proměnné Timer.

BindingPage.xaml (./DataBinding/)

```
namespace Concepts.DataBinding
{
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Concepts.DataBinding.BindingPage"
             Title="Data Binding"
             Padding="0,20,0,0">
    <ContentPage.Content>
        <StackLayout>
            <Label VerticalOptions="Start"
                   HorizontalOptions="CenterAndExpand"
                   Text="{Binding Timer}"/>
            <Label VerticalOptions="FillAndExpand"
                   VerticalTextAlignment="End"
                   HorizontalOptions="CenterAndExpand"
                   Text="{Binding Item.Nazev}"/>
            <Entry VerticalOptions="StartAndExpand"
                   x:Name="EntryPoznamka"   />
            <Button VerticalOptions="StartAndExpand"
                    Text="Ulož data"
                    Clicked="Uloz_clicked"/>
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

XAML soubor reprezentující View – konkrétně prvek Page(viz. 3.4.5.1). Na tomto příkladu lze demonstrovat, jak je svázán XAML soubor s code-behind: spojení je provedeno přes atribut **x:Class** , který musí být svou absolutní cestou stejný jako kódová třída. Obsahem stránky je StackLayout(viz. 3.4.5.2), který v sobě má 4 ovládací prvky: 2 popisky, vstupní pole a tlačítko. DataBinding je u obou popisků a u vstupního pole (pomocí code-behind). Atribut **x:Name** u ovládacího prvku **Entry**, slouží k identifikaci v code-behind. Binding se provádí nad Bindable propertami[5] - klíčovým slovem **Binding** a následným uvedením absolutní cesty k Property, ke které bude navázána. V tomto případě k vlastnosti ***Timer*** a ***Item.Nazev*** na ViewModelu reakci na změny zařídí mechanismy XAMLu.

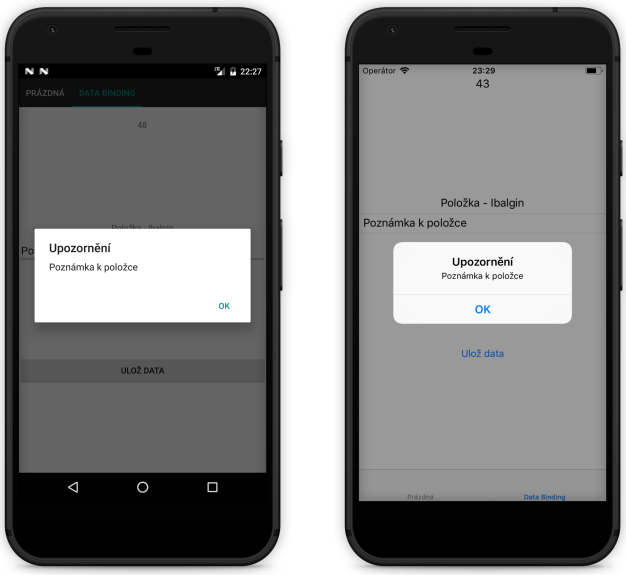
BindingPage.xaml.cs (./DataBinding/)

```
namespace Concepts.DataBinding
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
        public partial class BindingPage : ContentPage
        {
```

[5]Speciální vlastnost třídy, která může být navázána na konkrétní instanci ViewModelu

```
            public BindingPage ()
            {
                InitializeComponent ();
        BindingContext = new BindingViewModel();
            }

        protected override void OnBindingContextChanged()
        {
            base.OnBindingContextChanged();
            EntryPoznamka.SetBinding(Entry.TextProperty,
                "Item.Poznamka");
        }

        private void Uloz_clicked(object sender, ClickedEventArgs args)
        {
            var context = BindingContext as BindingViewModel;
            DisplayAlert("Upozornění", \$"{context.Item.Poznamka}",
                "OK");
        }
    }
}
```

Posledním souborem v tomto příkladu je code-behind pro třídu BindingPage, tedy opět vrstva View. V konstruktoru je do property **BindingContext**, což je reference View na ViewModel přiřazena nová instance třídy **BindingViewModel**, čímž dojte k vzájemnému propojení. Metoda **InitializeComponent** inicializuje strukturu View podle XAML souboru. Metoda **OnBindingContextChanged**, je volaná pokaždé, když dojde ke změně kontextu(tedy ViewModelu), druhý řádek této metody ilustruje nastavení data bindingu pomocí kódu (jedná se tedy o ekvivalent klíčového slova **Binding** v XAML souboru) – prvním parametrem je BindableProperty na ovládacím prvku a druhým opět absolutní cesta k vlastnosti na ViewModelu. Metoda **Uloz_clicked** se vyvolá při kliknutí na tlačítko uložit a zobrazí upozornění, jenž má jako text hodnotu zadanou jako poznámku ve vstupním poli obrazovky. Výsledná aplikace je vidět na obrázku 3.14.

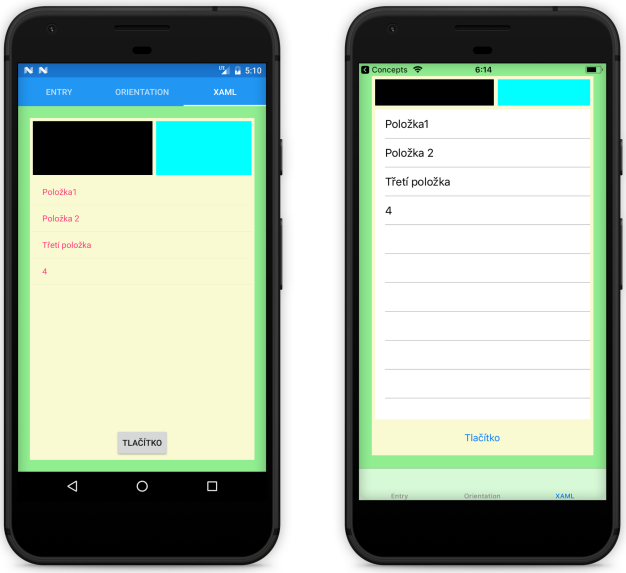


(a) (b)

Obrázek 3.14: Implementace data bindingu v projektu Concepts pro Android(a) a iOS(b)

3.4.5 GUI

GUI neboli grafické uživatelské rozhraní je specifické pro každou platformu. Framework Xamarin.Forms má vlastní strukturu, kterou lze variabilně vytvářet složitější hierarchické obrazovky,



(a) (b)

Obrázek 3.15: Ilustrativní příklad hierarchie obrazovky z kódu v XAML sekci pro Android(a) a iOS(b)

šablony pro zobrazení kolekcí a nebo vlastní ovládací prky.

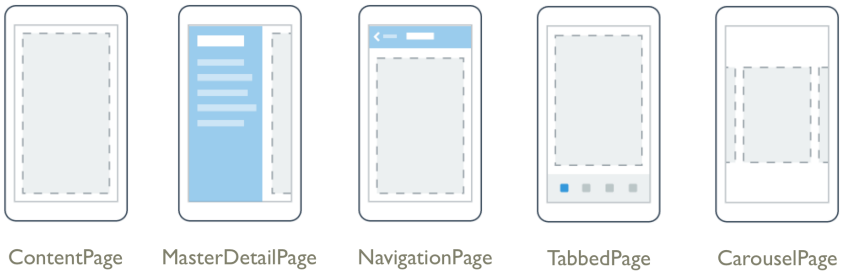
3.4.5.1 Page

Základním blokem každé obrazovky je ovládací prvek **Page**. Page je kontejner zabalující veškerý obsah. V Xamarin.Forms existují dva typy tohoto ovládacího prvku – TemplatedPage a MultiPage. Xamarin.Forms obsahuje předdefinované datové typy, které vystačí ve většině případů standardních aplikací (obrázek 3.16)

TemplatedPage
TemplatedPage je konečná obrazovka, která má zobrazovat full-screen obsah s definovanou šablonou. Implementace TemplatedPage, která je již součástí frameworku je ContentPage. Má proměnnou **Content**, která může být naplněna pouze jednou instancí typu View. Vedle toho je možné vytvořit libovolnou další TemplatedPage.

MultiPage
MultiPage je abstraktní datový typ, který definuje, že daná Page obsahuje jeden až několik potomků typu Page. Rozložení a přechod mezi jednotlivými stránkami v u MultiPage může být různorodé. Základní implementace ve frameworku pro třídu MultiPage jsou čtyři:

- **MasterDetailPage** – Obsahuje dvě Page, kde jedna je hlavní (**Detail**) a druhá je vyjížděcí menu z levé strany displeje (**Master**)
- **NavigationPage** – Potomci NavigationPage jsou uchovány jako v zásobníku, kde každá nová stránka přibyde na vrchol, a při navigaci zpět je stránka uvolněna. Zároveň obsahuje jednu tzv. **root** Page, kterou nelze uvolnit.
- **TabbedPage** – Stránky jsou umístěny vedle sebe jako záložky, přechod mezi nimi se dělá přes navigační lištu (pomocí jejich titulku a nebo ikonky).
- **CarouselPage** – Stránky jsou umístěny paralelně vedle sebe. Přechod mezi nimi je možný pomocí gesta posuvu vlevo či vpravo.



Obrázek 3.16: Přehled předdefinovaných Page a jejich grafické znázornění [24]

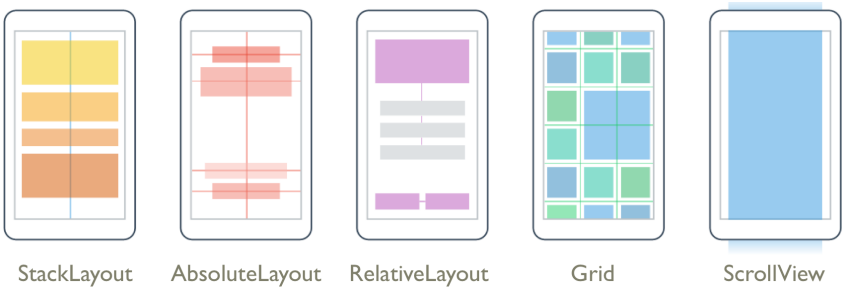
3.4.5.2 View

View je obecný grafický prvek, který lze umístit do Page. Každé View v základních knihovnách reprezentuje koncept, který lze chápat stejně na obou platformách (tlačítko, vstupní pole, seznam atd.), každý jiný jedinečný prvek pro jednu platformu musí být doimplementován explicitně (viz. kapitola 3.4.5.3).

Layout
Layout je stavební blok, který umožňuje skládat další ovládací prvky (View) strukturovaně na obrazovku bez ohledu na rozměry a orientaci displeje. Layout obsahuje klíčovou vlastnost **Children**, což je kolekce všech potomků. Framework obsahuje pět předdefinovaných Layout bloků, které lze využít:

- **StackLayout** – Obsahuje potomky umístěné vertikálně nebo horizontálně vedle sebe.
- **ScrollView** – Umožňuje posouvat se mimo hranice jedné obrazovky, která se nevejde celá na displej.

- **RelativeLayout** – Umožňuje umisťovat prvky v závislosti na pozici jejich rodičovských elementů.
- **AbsoluteLayout** – Používá umisťování prvků přesně na základě x,y koordinátů.
- **Grid** – Umožňuje vytvořit mřížku/tabulku a zarovnat prvky do řádků a sloupců



Obrázek 3.17: Předdefinované varianty Layout View
[25]

Control
Další vizuální prvky jsou opět zobecněné koncepty mobilních aplikací, které lze najít jak na Android tak na iOS. Některé předdefinované ovládací prvky jsou:

- **ListView** – Ovládací prvek pro zobrazení scrollovatelného seznamu dat v jednotlivých řádcích. Každý řádek odpovídá jedné instanci v proměnné **ItemsSource**. U každé položky lze navíc definovat její vzhled pomocí proměnné **ItemTemplate** na tomto prvku.
- **TableView** – Ovládací prvek podobný ListView, ovšem bez generické vlastnosti pro kolekci. TableView obsahuje kolekci Cell buněk rozdílných typů.
- **Button** – Ovládací prvek tlačítko pro vyvolání akce.
- **Label** – Zobrazení needitovatelného textového popisku.
- **Entry** – Vstupní pole pro zadávání textu. Zadávání lze omezit povolenými vstupními znaky(Zobrazovanou klávesnicí) nebo skrývat znak hvězdičkou pro tajná data
- **WebView** – Ovládací prvek využívající nativní webový prohlížeč dané platformy, který může zobrazovat jak webové stránky tak lokální soubory a generované HTML řetězce

Cell
Další důležitý blok je Cell. Jedná se o grafickou šablonu pro buňku (řádek) **ListView** nebo **TableView**. Každá buňka se při vykreslení dotáže na svoje datové property. Stejně jako ostatní prvky mají buňky BindableProperty, které lze svázat s konkrétní propertou ViewModelu. Framework obsahuje 5 zabudovaných typů zděděné od Cell:

- **Text Cell** – Slouží pro buňku s textovým titulkem a přidruženou hodnotou.
- **SwitchCell** – Slouží pro zachycení true/false hodnoty s textovým popiskem a vizuálním přepínačem
- **ImageCell** – Zobrazení ikonky v řádku.
- **EntryCell** – Slouží pro zadání textu s textovým popiskem.
- **ViewCell** – Slouží pro vytvoření vlastního vzhledu buňky pomocí property View, která způsobí, že buňka může být vykreslena se složitější hierarchií

Data Template
U ovládacího prvku **ListView** je potřeba definovat pro každou buňku v seznamu šablonu pomocí property **ItemTemplate** která je datového typu DataTemplate. DataTemplate je wrapper, který v sobě obsahuje informaci o konkrétním typu Cell kterou zobrazuje.Při vykreslení na displej se

instancuje nová buňka uloženého typu s kontextem zobrazované položky.

.NET Standard projekt (Concepts)
Item.cs (./DataTemplates/)

```
namespace Concepts.DataTemplates
{
    public class Item
    {
        public string Text { get; set; }
    }
}
```

Modelová třída s jednou property Text.

CustomCell.cs (./DataTemplates/)

```
<?xml version="1.0" encoding="UTF-8"?>
<ViewCell xmlns="http://xamarin.com/schemas/2014/forms"
          xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
          x:Class="Concepts.DataTemplates.CustomCell">
    <ViewCell.View>
        <StackLayout Orientation="Horizontal">
            <Label Text="{Binding Text}" FontSize="Large" />
            <Button Text="Přidat" HorizontalOptions="EndAndExpand" />
        </StackLayout>
    </ViewCell.View>
</ViewCell>
```

Vlastní implementace buňky pro ListView (potomek třídy ViewCell). Buňka obsahuje StackLayout se dvěma View – Label s bindingem na vlastnost Text na svázaném objektu a Button, pro ilustraci vykreslení vlastní buňky.

DataTemplateViewModel.cs (./DataTemplates/)

```
using System.Collections.ObjectModel;

namespace Concepts.DataTemplates
{
    public class DataTemplateViewModel
    {
        public ObservableCollection<Item> Items { get; set; }

        public DataTemplateViewModel()
        {
            Items = new ObservableCollection<Item>();
            Items.Add(new Item() { Text = "Položka 1" });
            Items.Add(new Item() { Text = "Položka 2" });
            Items.Add(new Item() { Text = "Položka 3" });
        }
    }
}
```

ViewModel pro přidruženou **Page**, s jednou vlastností kolekce objektů typu **Item**. Pro názornost je kolekce rovnou naplněna položkami, v komplexním příkladu by se položky stahovaly z databáze nebo pomocí webové služby.

DataTemplatesPage.xaml.cs (./DataTemplates/)

```
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace Concepts.DataTemplates
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
        public partial class DataTemplatesPage : ContentPage
```

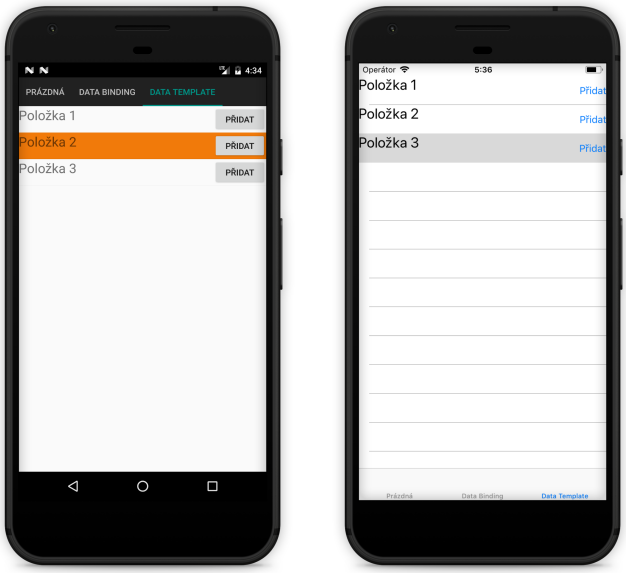
```
        {
                public DataTemplatesPage ()
                {
                        InitializeComponent ();
                        BindingContext = new DataTemplateViewModel();
                }
        }
}
```

Code-behind pro **Page**, nastavení kontextu na nový ViewModel.

DataTemplatesPage.xaml (./DataTemplates/)

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Concepts.DataTemplates.DataTemplatesPage"
             xmlns:dataTemplate="clr-namespace:Concepts.DataTemplates"
             Title="Data Template">
    <ContentPage.Content>
        <ListView HorizontalOptions="FillAndExpand"
                  VerticalOptions="FillAndExpand"
                  ItemsSource="{Binding Items}">
            <ListView.ItemTemplate>
                <DataTemplate>
                    <dataTemplate:CustomCell/>
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
    </ContentPage.Content>
</ContentPage>
```

Page s přidruženým **ListView**. V hlavičce souboru je atribut **xmlns:dataTemplate**, který definuje asociaci pro namespace pro využití v tomto souboru. Všechny třídy z namespace **Concepts.DataTemplates** lze nyní v XAML souboru instanciovat pomocí uvedení **dataTemplate:** a vlastního názvu třídy. ListView má property ItemTemplate, která je zde nastavena právě na instanciovanou třídu **CustomCell**. Zároveň je data bindingem svázána property **Items** z ViewModelu do Bindable property **ItemsSource**. Všechny položky v této kolekci se nyní vykreslí s přednastavenou šablonou. Výsledek je viditelný na obrázku 3.19



(a) (b)

Obrázek 3.18: Implementace Data template v projektu Concepts pro Android(a) a iOS(b)

3.4.5.3 Custom renderer

Občas se stejný prvek zobrazuje na různých platformách rozdílně, ať už se jedná o rozdílnou konvenci platformy nebo složitější ovládací prvky. V takovém případě je potřeba korekce na straně specifických platformních projektů – nejde to na úrovni sdíleného kódu. Pro konkrétní platformu vlastnosti při generování vizuálních prvků se využívá mechanismus Custom rendereru.

Portable projekt (EreceptSamples(Portable))
Item.cs (CustomEntry.cs (./CustomControls/)

```
using Xamarin.Forms;

namespace EreceptSamples.CustomControls
{
    public class CustomEntry : Entry
    {

        public Color TextFieldColor
        {
            get { return (Color)GetValue(MyPropertyProperty); }
            set { SetValue(MyPropertyProperty, value); }
        }

        public static readonly BindableProperty MyPropertyProperty =
            BindableProperty.Create(nameof(TextFieldColor),
            typeof(Color), typeof(CustomEntry), Color.White);
    }
}
```

Ve společném projektu je vlastní ovládací prvek definován jako potomek jiného již existujícího – některého z View (samotné View lze také použít jako předka vlastního prvku). V příkladě výše je definováno vlastní zadávací pole, které má oproti výchozímu poli novou Bindable property **TextFieldColor**, pro dynamické nastavení barvy pozadí.

EntryPage.xaml (./CustomControls/)

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
```

```
                xmlns:controls="clr-namespace:EreceptSamples.CustomControls"
                x:Class="EreceptSamples.CustomControls.EntryPage"
                Title="Entry">
        <StackLayout Padding="10">
            <!-- Základní entry -->
            <Label Text="Základní Entry" />
            <Entry />
            <!-- Vlastní entry -->
            <Label Text="Vlastní Entry" />
            <controls:CustomEntry TextFieldColor="LightGreen" />
            <controls:CustomEntry TextFieldColor="LightPink" />
            <controls:CustomEntry TextFieldColor="LightYellow" />
        </StackLayout>
    </ContentPage>
```

Vlastní Page, která obsahuje 4 viditelné ovládací prvky, z toho 3 vlastní zadávací pole s vyplněnou barvou pozadí. Jako již v předchzích příkladech, je potřeba pro vlastní třídy definovat v atributu kořenového elementu namespace **xmlns:controls** pro využití tříd z něj.

Android projekt (EreceptSamples.Android)
CustomEntryRenderer.cs (./CustomRenderers/)

```
[assembly: ExportRenderer(typeof(CustomEntry),
    typeof(CustomEntryRenderer))]
namespace EreceptSamples.Droid.CustomRenderers
{
    public class CustomEntryRenderer : EntryRenderer
    {
        protected override void
            OnElementChanged(ElementChangedEventArgs<Entry> e)
        {
            base.OnElementChanged(e);

            var element = e.NewElement as CustomEntry;

            if(element != null)
            {
                Control?.SetBackgroundColor(Android.Graphics.Color.Transparent);

                var backgroundDrawable = new GradientDrawable();
                backgroundDrawable.SetCornerRadius(15.0f);
                backgroundDrawable.SetColor(element.TextFieldColor.ToAndroid());
                backgroundDrawable.SetStroke(1, Color.Black.ToAndroid());
                Control?.SetBackground(backgroundDrawable);
            }
        }
    }
}
```

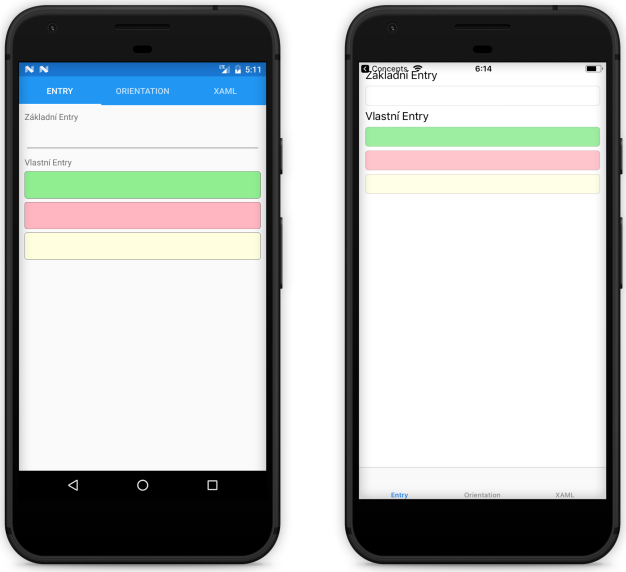
Třída vlastního rendereru dědí od již definovaných rendererů (V případě příkladu **EntryRendereru**, název vlastní třídy je libovolný, ale konvence je název vlastního ovládacího prvku s postfixem Renderer. Xamarin pozná, že jde o vlastní vykreslování díky atributu **ExportRenderer**, nad třídou, která má dva parametry – typ vlastního ovládacího prvku a typ rendereru. Za běhu pokud se má vykreslit ovládací prvek je vyhledaná instance rendereru a použity přepsané metody.

iOS projekt (EreceptSamples.iOS)
CustomEntryRenderer.cs (./CustomRenderers/)

```
[assembly: ExportRenderer(typeof(CustomEntry),
    typeof(CustomEntryRenderer))]
namespace EreceptSamples.iOS.CustomRenderers
{
    public class CustomEntryRenderer : EntryRenderer
    {
        protected override void
            OnElementChanged(ElementChangedEventArgs<Entry> e)
```

```
            {
                base.OnElementChanged(e);

                var element = e.NewElement as CustomEntry;

                if(element != null)
                {
                    if (Control != null)
                    {
                        Control.BackgroundColor =
                            Color.Transparent.ToUIColor();
                        Control.Layer.CornerRadius = 5.0f;
                        Control.Layer.BackgroundColor =
                            element.TextFieldColor.ToCGColor();
                        Control.Layer.BorderColor =
                            Color.DarkGray.ToCGColor();
                        Control.Layer.BorderWidth = 0.2f;
                    }
                }
            }
        }
    }
```

Vlastní renderer na platformě iOS má úplně stejné rozhraní jako výše zmíněný renderer na Android. Jediné, čím se liší obě implementace je datový typ property **Control**, která je určena na základě rodičovské třídy (V tomto případě EntryRenderer má property Control typu **UIKit.UITextField** pro iOS, resp. **Android.Widget.EditText** v případě Androidu, což jsou zrcadlené třídy z nativních API.

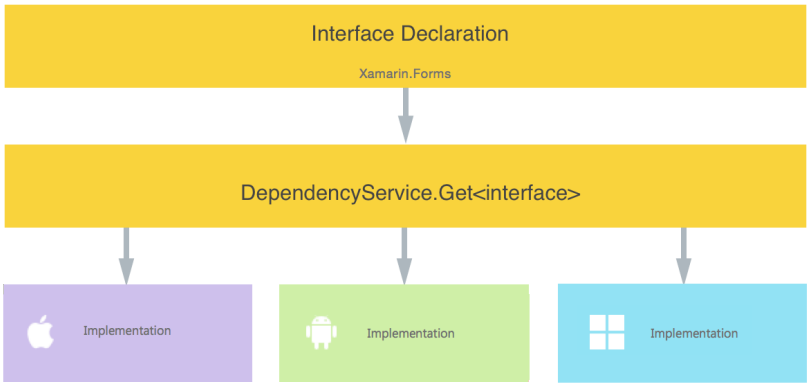


(a) (b)

Obrázek 3.19: Vykreslení vlastního ovládacího prvku Custom renderery pro Android(a) a iOS(b)

3.4.6 Dependency Injection

V některých případech není funkcionalita konkrétní platformy pokryta základními knihovnami Xamarin.Forms. Stejně jako u implicitních knihoven i u nově navržených tříd je ve sdíleném projektu vystaveno pouze rozhraní dané třídy a konkrétní implementace je v koncových projektech, jež se při spuštění aplikace registruje. Dále pak stačí používat rozhraní, které si drží referenci na konkrétní implementaci. Vhodné pro – práce se soubory, nativní API jako navigace, mapy, fotoaparát apod. Spousta funkcionalit je takovým způsobem navíc dostupných jako opensource Nuget balíčky (zásuvné moduly do projektu). Koncept Dependency Injection je v Xamarin.Forms realizován pomocí třídy DependencyService viz. 3.20.



Obrázek 3.20: Koncept Dependency Injection v Xamarin.Forms pomocí DependencyService [26]

Portable projekt (EreceptSamples(Portable))
IDeviceOrientation.cs (./DependencyServices/)

```
namespace EreceptSamples.DependencyServices
{
    public enum DeviceOrientations
    {
        Undefined,
        Landscape,
        Portrait
    }

    public interface IDeviceOrientation
    {
        DeviceOrientations GetOrientation();
    }
}
```

Definice rozhraní **IDeviceOrientation** ve společném projektu, které definuje metodu **GetOrientation** pro zjištění rotace displeje.

DeviceOrientationPage.xaml.cs (./DependencyServices/)

```
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace EreceptSamples.DependencyServices
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class DeviceOrientationPage : ContentPage
    {
        public DeviceOrientationPage()
        {
            InitializeComponent();
```

```
        }

        private void Button_Clicked(object sender, System.EventArgs e)
        {
            var deviceOrientation =
                DependencyService.Get<IDeviceOrientation>();
            OrientationLabel.Text =
                deviceOrientation.GetOrientation().ToString();
        }
    }
}
```

Page, která obsahuje pouze tlačíko pro zjištění orientace – metoda **Button_Clicked**. V metodě se konkrétní implementace definovaného rozhraní získá generickou metodou **Get** nad statickou třídou **DependencyService**.

Android projekt (EreceptSamples.Android)
DeviceOrientation.cs (./DependencyServices/)

```
[assembly:
    Xamarin.Forms.Dependency(typeof(DeviceOrientationImplementation))]
namespace EreceptSamples.Droid.DependencyServices
{
    public class DeviceOrientationImplementation : IDeviceOrientation
    {
        public DeviceOrientationImplementation() { }

        public static void Init() { }

        public DeviceOrientations GetOrientation()
        {
            IWindowManager windowManager =
                Android.App.Application.Context
            .GetSystemService(Context.WindowService).JavaCast<IWindowManager>();
            var rotation = windowManager.DefaultDisplay.Rotation;
            bool isLandscape = rotation == SurfaceOrientation.Rotation90
                || rotation == SurfaceOrientation.Rotation270;
            return isLandscape ? DeviceOrientations.Landscape :
                DeviceOrientations.Portrait;
        }
    }
}
```

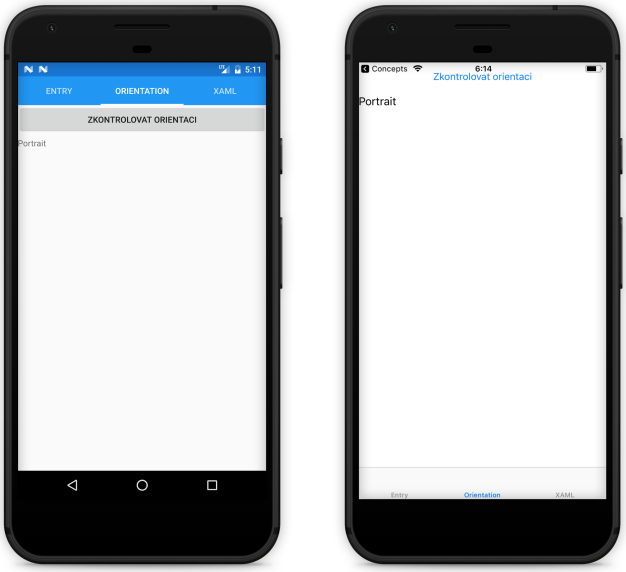
Registrace rozhraní pro následné využití v metodě Get se využije atribut **Dependency** nad třídou s jedním parametrem – typem implementace, která dědí od implementovaného rozhraní. Metoda **GetOrientation** pak zjistí orientaci displeje na Android zařízeních pomocí nativního API.

iOS projekt (EreceptSamples.iOS)
DeviceOrientation.cs (./DependencyServices/)

```
[assembly:
    Xamarin.Forms.Dependency(typeof(DeviceOrientationImplementation))]
namespace EreceptSamples.iOS.DependencyServices
{
    public class DeviceOrientationImplementation : IDeviceOrientation
    {
        public DeviceOrientationImplementation() { }

        public DeviceOrientations GetOrientation()
        {
            var currentOrientation =
                UIApplication.SharedApplication.StatusBarOrientation;
            bool isPortrait = currentOrientation ==
                UIInterfaceOrientation.Portrait
                || currentOrientation ==
                    UIInterfaceOrientation.PortraitUpsideDown;
```

```
                return isPortrait ? DeviceOrientations.Portrait :
                    DeviceOrientations.Landscape;
            }
        }
    }
```

Ekvivalentně se implementuje rozhraní v iOS projektu. Zde je opět rozdílná pouze vlastní implementace pro zjištění orientace – registrace třídy se dělá totožně jako v případě Androidu.



(a) (b)

Obrázek 3.21: Zjištění orientace displeje pomocí Dependency Injection pro Android(a) a iOS(b)

Kapitola 4

eRecept

Plán elektronizace zdravotnictví začal již v roce 2013. Vláda ČR dne 7. července 2016 schválila a MZ ČR zveřejnilo strategii kompletního přechodu z papírového na elektronické zdravotnictví, které probíhá v souladu s principy eGovermentu (viz. 4.2, který je plně v kompetenci Ministerstva vnitra ČR.

Elektronicky se budou moci například objednávat zdravotní služby,pro pacienty bude snadný přístup k osobnímu zdravotnímu záznamu a léčebným plánům, řeší se elektronická neschopenka, která bude přínosem jak pro lékaře tak pacienty.

V oblasti elektronizace zdravotnictví zaujímá nyní Česká republika jedno z posledních míst v Evropě. Po pěti letém přechodném období od 1.1.2018 bude elektronické předepisování (e Recept) povinné pro všechny lékaře, lékárny a zdravotní zařízení.

4.1 Zákonné úpravy

Vzhledem k rozsáhlosti systému musí eRecept splňovat celou řadu zákonných a dalších norem:

- **Etický kodex v bodě 6: „Práva pacientů“** - Zde se mimo jiné řeší to, že každý pacient má právo na to, aby veškeré jím poskytnuté infomace, veškeré záznamy a zprávy měly důvěrný charakter i v případě zpracování na počítači.
- **Listina základních práv a svobod, článek 10, bod 3** - Řeší se základní právo pacienta na ochranu před zneužíváním údajů.
- **Stavovský předpis č. 10, §2** - Tento předpis řeší povinnosti lékařů vést evidenci o pacientovi tak, aby byla přiměřená ochrana dokumentace, nemohlo dojít k její změně, zneužití nebo zničení.
- **Evropská charta práv pacientů** - Tato listina zajišťuje každému právo na ochranu osobních dat, včetně údajů o zdravotním stavu, léčebných metodách, právo na soukromý během vyšetření a lékařských zákroků všeobecně.
- **Zákon o zdravotních službách** - Definuje právo občanů na respektování soukromí při poskytování těchto služeb
- **Zákon o ochraně osobních údajů** - specifikuje situace, kdy je možno zpracovávat citlivé údaje, jakým způsobem postupovat, aby každý, kdo s daty pracuje zabezpečil to, aby subjekt údajů neutrměl újmu na svých právech, zejména zachování lidské důstojnosti, aby neměl zásah do svého soukromého a osobního života.
- **Data Protection Directive** - směrnice Evropského parlamentu a rady o ochraně fyzických osob v souvislosti se zpracováním osobních údajů a o volném pohybu těchto údajů.
- **Vyhláška o zdravotnické dokumentaci** - Řeší způsob, jakým musí technické prostředky pro vedení zdravotnické dokumentace zabezpečit výpočetní techniku, zamezit přístupu neoprávněných osob k této dokumentaci, vedení evidence všech přístupů k ní, včetně způsobu, jakým způsobem se budou provádět opravy, mazání a změny.

- **Trestní zákoník** - Určuje postih pro toho, kdo ať třeba i z nedbalosti poruší zákonem uloženou povinnost mlčenlivosti, prozradí nebo umožní dalším osobám přístup k údajům získaným v souvislosti s výkonem svého povolání, funkce nebo zaměstnání, a tak způsobí vážnou újmu tomu, koho se osobní údaje týkají.

4.2 eGovernment

Účelem eGovernmentu je zlepšení fungování veřejné správy a jejího vztahu k veřejnosti s využitím informačních technologií. Slouží k výměnám informací s občany, soukromými organizacemi i jinými veřejnými institucemi a očekává se od něj také poskytování rychlých, dostupných a kvalitních informačních služeb. Měl by tak především usnadnit styk veřejnosti s úřady[27]. Hlavní výhody elektronizace státní správy jsou tyto:

- Snížení počtu pracovníků a snížení nákladů
- Zrychlení a zjednodušení administrativních procesů
- Vyšší spokojenost konzumentů služby a jejich větší pohodlí
- Snížení chybovosti poskytnutých dat
- Flexibilita (služba může být poskytnuta i mimo pracovní hodiny instituce)

Koncept eGovernmentu stojí na třech základních pilířích:

- **Jednotné informace** – Občan by měl registraci k tzv. Základním registrům (Zákon 111/2009 Sb.) provést pouze jednou. Veškeré informační systémy by pak měly identitu občana sdílet. Základní registry jsou čtveřice centrálních míst, které obsahují data o občanech.
- **Kontaktní místo** – Občan by měl mít možnost komunikace se státem přes jedno kontaktní místo pro vyřízení všech potřebných záležitostí. V České republice je zprovozněna síť Czech Point (České Podací Ověřovací Informační Národní Terminály), a se státní správou lze komunikovat prostřednictvím datové schránky.
- **Proaktivní přístup** - Většina agendy spojená s jednou žádostí by měla být úřady vyřízena bez další nutné činnosti občana – tedy stav, kdy obíhají doklady mezi jednotlivými úřady. Občan má být pouze informován o tom, v jakém stavu jeho žádost momentálně je a jak je nakládáno s jeho osobními údaji. Komunikaci mezi registry a jednotlivými orgány státní správy[28] zajišťuje Informační systém základních registrů (ISZR).

4.2.1 eIdentita

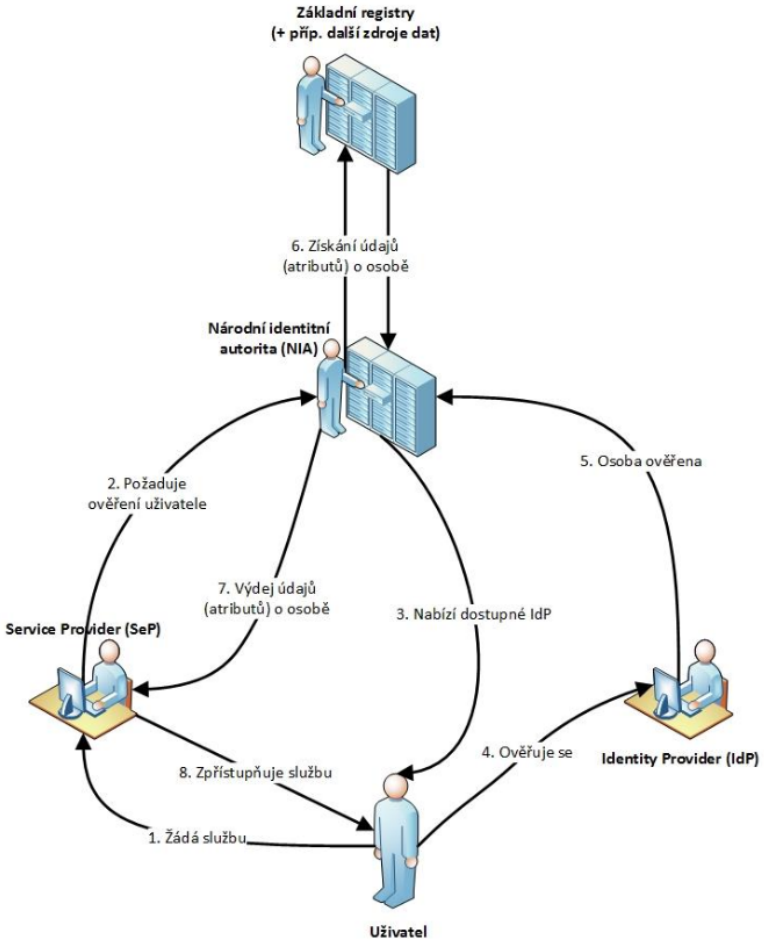
Elektronická identita(eIdentita) umožňuje mnohem rychleji vyřizovat na úřadech své požadavky, občan nemusí opakovaně vyplňovat své identifikační údaje a zároveň má přehled o údajích, které o nich vede stát. Tyto údaje jsou v rámci státní správy a samosprávy sdíleny. Data jsou uložena na jednom místě, ze kterého se jakékoliv změny automaticky aktualizují do ostatních systémů. Identifikační bránou pro jednoduchý a bezpečný přístup k online službám veřejné správy je eIdentita.cz.

4.2.2 Národní identitní autorita (NIA)

Zákonným prostředkem (v souladu s mezinárodní legislativou eIDAS) pro správu identit občanů je tzv. Národní identitní autorita (NIA), která zprostředkovává služby důvěryhodných poskytovatelů identit (Identity provider - IdP) jednotlivým důvěryhodným poskytovatelům služeb (Service Provider - SeP) vyžadujícím důvěryhodnost autentizací přistupujících subjektů (uživatelů)[29]. Mezi Service Providery patří poskytovatelé veřejných služeb, Identity provideři jsou například základní registry. NIA poskytuje rozhraní pro registraci jednotlilvých Identity Providerů a Service providerů nebo rozhraní pro uživatelské nastavení zpřístupnění subjektem definovaných osobních údajů ostatním systémům.

Service provider tak nemusí mít vlastní databázi uživatelů, ale čerpá informace o připojeném uživateli z registru na základě předem sdílených informací. Díky tomu je zajištěna vyšší bezpečnost informačních systémů, jedny přihlašovací údaje na straně uživatele k různým veřejným službám a oddělení vlastní agendy od správy uživatelských účtů. Celý proces práce mezi NIA, IdP, SeP

a uživatelem je znázorněn na obrázku 4.1. Mezi kroky 1, 3 a 5 dochází k přesměrování na cílový informační systém.



Obrázek 4.1: Proces komunikace se Service Providerem prostřednictvím NIA [29]

4.3 eRecept

Jedna podmnožina elektronizace státní správy, která se nazývá eHealth se zabývá integrací moderních technologií do českého zdravotnictví. V roce 2007 byl přijat zákonný předpis č.378/2007 Sb., který nařizuje Státnímu ústavu pro kontrolu léčiv (dále jen **SÚKL**) vybudování Centrálního úložiště elektronických receptů (**CÚeR**) - viz. 4.3.2, což je komplexní řešení evidence léčiv vydaných pacientovi pomocí automatických online hlášení odesílaných přímo z lékárny[30].

4.3.1 Obecný princip eReceptu

Elektronický recept(dále **eRecept**), je lékařský předpis, který je vydán v elektronické podobě ,prostřednictvím specializovaného software, do Centrálního úložiště. Každý eRecept, který je jedinečný svým unikátním identifikátorem, je přístupný konkrétnímu pacientovi. Po předložení identifikátoru je možné na něj nahlížet lékařem a lékárníkem.

Pro lékaře je od 1. ledna 2018 povinné vydávání elektronických receptů pro všechny pacienty bez výjimky. Pro pacienta samotného existují 4 možnosti (Zvláštním případem je vydávání klasického papírového receptu bez identifikátoru, v zákonem definovaných výjimečných případech), jakými se jedinečný identifikátor svého předpisu, v případě úspěšného založení v CÚeR, může zdarma dozvědět:

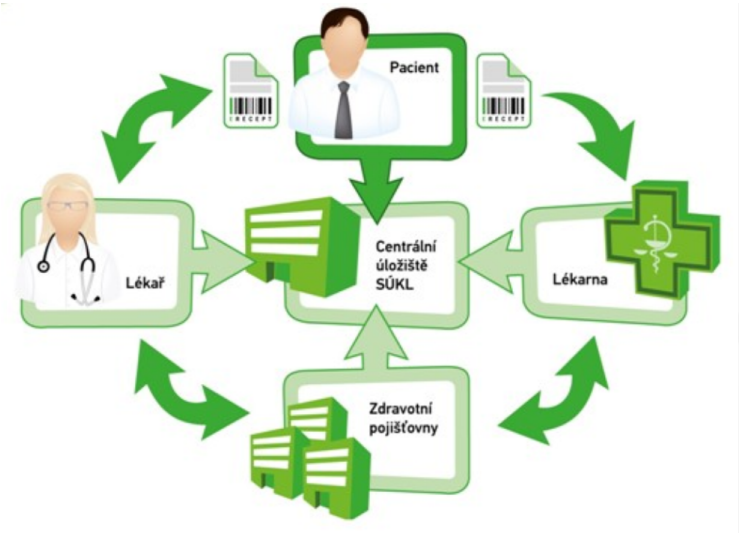
- Pomocí papírové průvodky vytištěné lékařem (obrázek 4.2)

- Prostřednictvím SMS
- Prostřednictvím e-mailu
- Prostřednictvím webové a mobilní aplikace

Obrázek 4.2: Vzorová průvodka eReceptu včetně identifikátoru v podobě čárového kódu [31]

Na základě předloženého identifikátoru může lékárník načíst eRecept v lékárenském systému, a nahlížet na pacientův předpis, případně vydat léčivé přípravky na něm uvedené. Lékárník má dále možnost v případě nedostupnosti léků vydat pouze část přípravků, případně vydat lék se stejnou léčivou látkou. Jakmile dojde k vydání léků pacientovi a promítnou se změny v CÚeR, může si předepisující lékař zkontrolovat, že si jeho pacient léky vyzvedl.

Systém dále umožňuje nahlížení zdravotních pojišťoven na statistická data z CÚeR, případně kontrolu lékařů a lékáren vzhledem k vypisovaným a vydávaným receptům. Vzájemné propojení všech aktérů je zřejmé z obrázku 4.3.



Obrázek 4.3: Diagram zapojení jednotllivých aktérů do systému eReceptu [32]

4.3.1.1 Výhody eReceptu

Koncept eReceptu přináší mnoho výhod pro všechny zainteresované strany.

Výhody v době spuštění

- eRecept není možné zfalšovat nebo vydat léčivý přípravek bez předpisu, neboť k založení a výdeji je potřeba uživatelský účet a při zakládání se recept podepisuje jednoznačným certifikátem.
- Lékař má, v případě svých předepsaných receptů možnost si zkontrolovat, že si jeho pacienti léky vyzvedli, případně zda mu byly skutečně vydány předepsané léky.
- Lék může být pacientovi vydán vzdáleně, bez nutné osobní návštěvy lékaře, což ušetří návštěvy lékaře u opakovaných výdejů stejných léčiv.
- Lékaři mohou předepisovat léky i mimo svou ordinaci prostřednictvím webové a mobilní aplikace.
- Jedno úložiště zjednoduší práci Policie ČR a Zdravotních pojišťoven, kterým poskytne přehled o výdejích léků.

Výhody v budoucnu

- Pacient bude mít k dispozici kompletní lékový záznam, který mu umožní kontrolu, že mu nejsou předepisovány falešné recepty.
- Systém umožní více položek na jednom receptu (Aktuálně jsou legislativou povoleny 2 položky na recept).
- Pacient může svému ošetřujícímu lékaři zpřístupnit kompletní lékový záznam, pro zabránění duplicit léčivých látek a kontraindikací. Systém bude takové případy automaticky hlídat.

4.3.2 Centrální úložiště elektronických receptů (CÚeR)

Centrální úložiště receptů je systémem, který umožňuje rozšířit sledování léčiva na jeho cestě od výrobce k distributorovi dál do lékárny až ke koncovému pacientovi. Systém zvyšuje bezpečnost léčiv pro pacienty a omezuje zneužívání léčiv obsahujícího pseudoefedrin k výrobě drog. SÚKL může účinněji zakročit při ohrožení zdraví pacientů v ČR. Pacienti současně získají kompletní přehled svých léků přes internet a dojde i k celkovému zprůhlednění obchodu s léčivy na trhu v ČR.

Pro příjem dat z lékáren a zdravotnických zařízení byly připraveny konektory webových služeb. Při založení předpisu je mu přidělen unikátní identifikátor, který je předán v rámci potvrzení o přijetí zpět původnímu odesílateli. Kvalitu uložených dat zajišťuje několik úrovní kontrol přijímaných zpráv.

Každý elektronický recept musí být elektronicky podepsán a validita tohoto podpisu je ověřována systémem CÚeR. Veškerá odchozí data týkající se eReceptů jsou označena elektronickou značkou (tedy elektronicky podepsána serverem). Přístup pacienta ke svému lékovému účtu je možný také pouze s použitím zaručené elektronické identifikace (například pomocí NIA).

4.3.3 Přístup k CÚeR

Pro uživatelský přístup do Centrálního úložiště je potřeba vytvořit příslušné uživatelské účty, zaregistrovat se jako subjekt k přístupu do úložiště v případě lékařů a lékárníků a vygenerovat si přístupové certifikáty a přístupové údaje. SÚKL na svých webových stránkách poskytuje návody jak si jednotlivé účty pro pacienta[33], lékaře[34] a lékárníka[35] zprovoznit.

4.3.3.1 Jednotné rozhraní webových služeb

CÚeR poskytuje certifikovaným vývojářům veřejné rozhraní webových služeb pro komunikaci z jednotlivých aplikací. Webové služby běží na technologii SOAP skrze zabespečenou https komunikaci. Všechny vystavené služby jsou bezestavové, tudíž každý požadavek musí obsahovat autentikační údaje. Pro identifikaci uživatele je použit bezvýznamový identifikátor uživatele, který je přidělený SÚKL jako specifický pro každého uživatele. ID uživatele použité pro autentizaci se musí rovnat ID lékaře, lékárníka či jiného uživatele použité ve XML zprávě. Autentizace přistupujícího subjektu je prováděna ověřením předaného uživatelského jména a hesla při každém volání webových služeb.

Příklad těla zprávy pro načtení receptu a následné odpovědi:

Vzorové XML požadavku pro načtení receptu z CÚeR:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:v230="http://www.sukl.cz/erp/201704"
    xmlns:v2301="http://www.sukl.cz/erp/201704"
    xmlns:xd="http://www.w3.org/2000/09/xmldsig#">
  <soapenv:Header/>
  <soapenv:Body>
    <v230:NacteniPredpisuDotaz>
      <v2301:Doklad>
        <v2301:Pristupujici>
          <v2301:Uzivatel>7DBBDDD2-C28E-4372-BFBA-61E80CBF36F6</v2301:Uzivatel>
          <v2301:Pracoviste>00000000011</v2301:Pracoviste>
        </v2301:Pristupujici>
        <v2301:Identifikator>
          <v2301:ID_Dokladu>PCAFJKRD4NV3</v2301:ID_Dokladu>
        </v2301:Identifikator>
      </v2301:Doklad>
      <v2301:Zprava>
        <v2301:ID_Zpravy>b65964c3-bd06-4d11-8bb7-8796fb68afc4</v2301:ID_Zpravy>
        <v2301:Verze>201704A</v2301:Verze>
        <v2301:Odeslano>2017-05-22T01:37:06.507+02:00</v2301:Odeslano>
        <v2301:SW_Klienta>0123456789AB</v2301:SW_Klienta>
      </v2301:Zprava>
    </v230:NacteniPredpisuDotaz>
  </soapenv:Body>
</soapenv:Envelope>
```

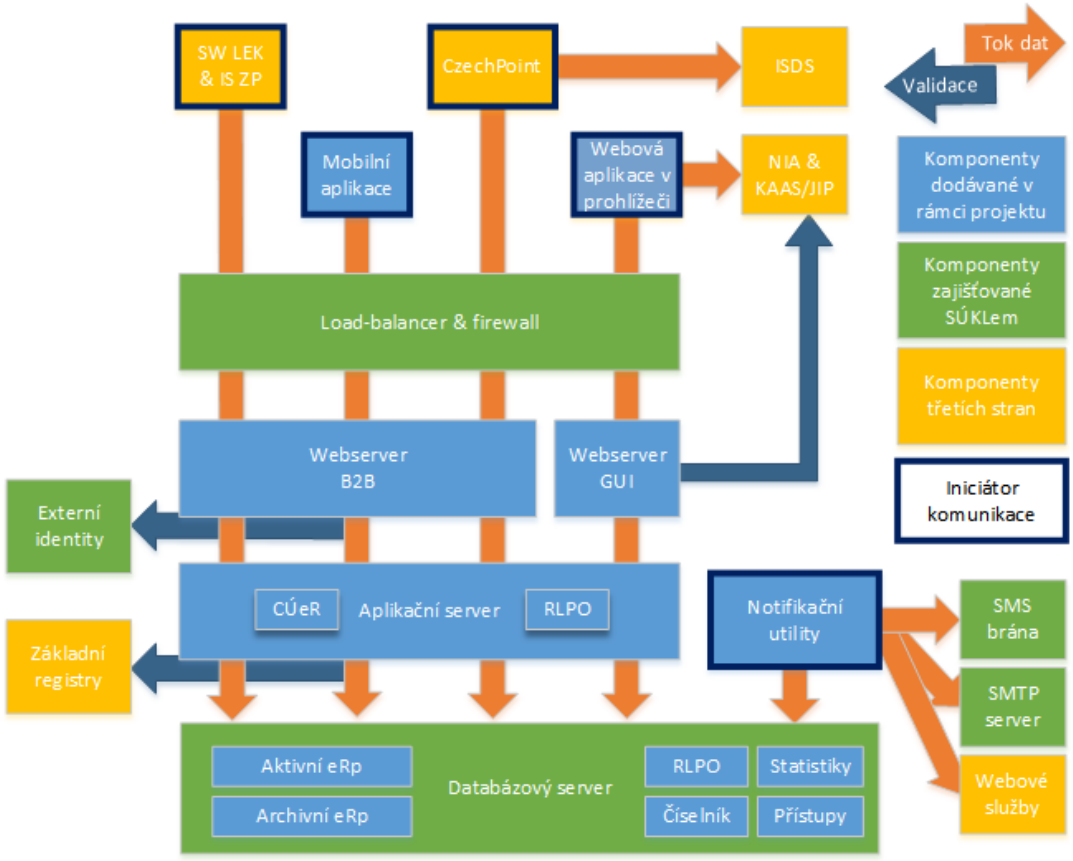
Vzorové XML (zkrácené) s odpovědí na požadavek načtení receptu z CÚeR:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header
      xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"/>
  <soap:Body>
    <erp:NacteniPredpisuOdpoved
        xmlns:erp="http://www.sukl.cz/erp/201704"
        xmlns:sig="http://www.w3.org/2000/09/xmldsig#">
      <erp:Doklad>
        <!-- ... -->
        <erp:Pacient>
          <erp:Totoznost>
            <erp:Jmeno>
              <erp:Prijmeni>MATOUŠEK</erp:Prijmeni>
              <erp:Jmena>JIŘÍ</erp:Jmena>
            </erp:Jmeno>
            <!-- ... -->
          </erp:Totoznost>
          <!-- ... -->
        </erp:Pacient>
        <erp:Predepisujici>
          <erp:Lekar>
```

```
            <erp:Kod>669CAC52-B591-4276-8815-F81A2F97192A</erp:Kod>
            <erp:Jmeno>
              <erp:Prijmeni>Novák</erp:Prijmeni>
              <erp:Jmena>Petr</erp:Jmena>
            </erp:Jmeno>
          </erp:Lekar>
          <erp:PZS>
            <erp:Kod>00000000007</erp:Kod>
            <erp:Nazev>Pokusný poskytovatel s.r.o.</erp:Nazev>
            <!-- ... -->
          </erp:PZS>
        </erp:Predepisujici>
        <erp:PLP>
          <!-- ... -->
        </erp:PLP>
        <erp:Vydej>
          <erp:ID_Dokladu>VCAXDUE5JNEA</erp:ID_Dokladu>
          <!-- ... -->
        </erp:Vydej>
      </erp:Doklad>
      <erp:Zprava>
        <erp:ID_Zpravy>AC62F34E-9040-4396-BDEA-8F209FE082A0</erp:ID_Zpravy>
        <erp:Verze>201704A</erp:Verze>
        <erp:Odeslano>2017-05-22T01:37:06.991+02:00</erp:Odeslano>
        <erp:ID_Podani>B65964C3-BD06-4D11-8BB7-8796FB68AFC4</erp:ID_Podani>
        <erp:Prijato>2017-05-22T01:37:06.949+02:00</erp:Prijato>
      </erp:Zprava>
    </erp:NacteniPredpisuOdpoved>
  </soap:Body>
</soap:Envelope>
```

4.3.3.2 Webová a mobilní aplikace SÚKL

Díky technologii SOAP je možné k CÚeR přistupovat pomocí volání webových služeb skrze HTTPS protokol z různých klientských aplikací. V rámci projektu eRecept realizovaném společností Solitea Business Solutions s.r.o. byly na základě požadavků ze strany SÚKL realizovány webové a mobilní klientské aplikace pro lékaře a pacienty (v průběhu implementace byla ještě objednána a vyvinuta i aplikace pro lékárny). Oba typy aplikací jsou SÚKLem veřejně a zdarma dostupné – o implementaci mobilních aplikací pojednává tato práce. Kompletní architektura tohoto řešení je zřejmá z obrázku 4.4.



Obrázek 4.4: Kompletní architektura řešení eRecept v rámci projektu Solitea Business Solutions s.r.o.

[36]

Kapitola 5

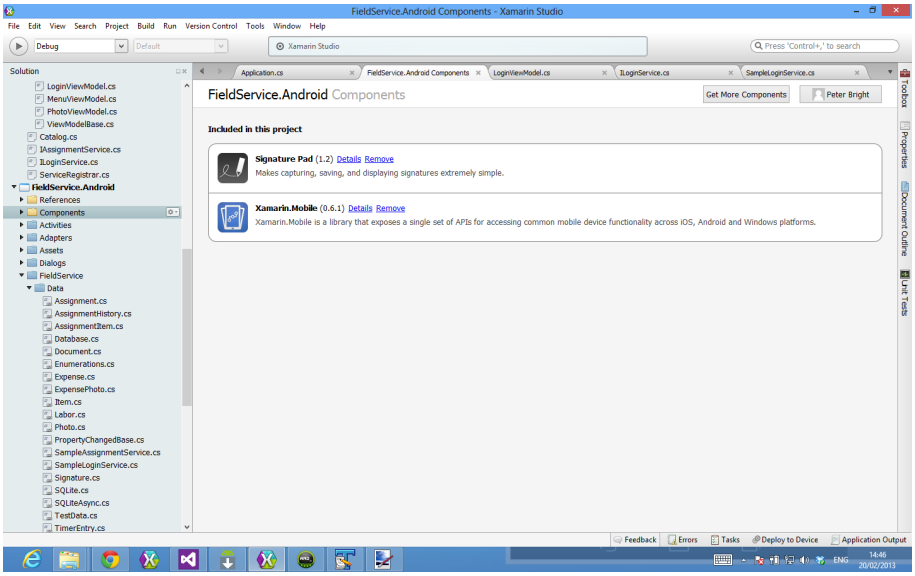
Vývojová prostředí a užitečné nástroje

Aplikace v Xamarinu může být vytvořena na libovolné platformě operačního systému, kde je nainstalovaný příslušný kompilátor. To znamená, že aplikace pro Windows mohou být psány na systémech Windows, a iOS/macOS aplikace naopak potřebují operační systém macOS. Aplikace pro Android mohou být kompilovány jak na zařízení s Windows, tak s macOS. Pomocí vývojového prostředí se dá docílit ulehčení práce s překladem kódu na konkrétní platformě, jelikož je vytvořen mezi dvěma počítači rozdílných platforem virtuální můstek, který provede všechny potřebné operace automaticky (jedná se jak o samotnou kompilaci, tak o možnost debuggování aplikace na vzdáleném připojení).

Autor tohoto textu nemá za cíl popsat jednotlivá prostředí ze všech úhlů pohledu, nicméně je dobré zmínit jejich klíčové vlastnosti, které byly zcela nebo částečně využity při vývoji reálné aplikace popsané v této práci. Zároveň se v této podkapitole probírá asi nejtěžší část konfigurace vývojového prostředí – propojení s Mac zařízením (kapitola **??**).

5.1 Xamarin Studio

Ještě před tím, než byl Xamarin koupen společností Microsoft, používalo se pro vývoj projektů IDE zvané Xamarin Studio, které bylo dostupné pro operační systémy Windows i macOS. Tato práce vznikla ale až po zmíněné akvizici, kdy již není možné Xamarin studio pro Windows stáhnout. Pro Mac zařízení je nicméně Xamarin studio stále alternativou pro vývoj, nicméně hlavní podpora se nyní zaměřuje na vývoj plnohodnotného IDE Visual Studio for Mac.



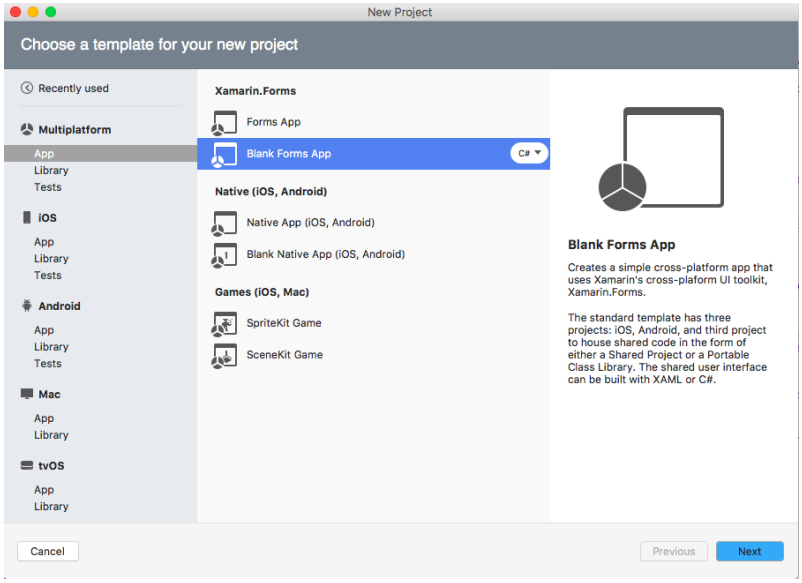
Obrázek 5.1: Vývojové prostředí Xamarin studio na platformě Windows [37]

5.2 Visual Studio for Mac

Visual Studio for Mac je IDE pro operační systém macOS. Je určeno pro vývoj projektů ve skupině jazyků .NET, zaměřené na projekty Xamarin (Xamarin.Forms, Xamarin.iOS, Xamarin.Android), nicméně mohou být vytvářeny i ostatní typy aplikací (konzolové, webové apod.). Bez poznámky nemůže být opomenut fakt, že hlavním designérem grafického ztvárnění tohoto vývojového prostředí je Čech Václav Vančura. Toto vývojové prostředí je de facto nadstavbou nad Xamarin Studiem se snahou graficky a uživatelsky sjednotit používání s Visual Studiem na operačním systému Windows. Na rozdíl od verze pro Windows ovšem nejde ve Visual Studio for Mac integrovat projekt s TFS[6], což může být problém při práci ve větším týmu. Nabízená je pouze varianta verzování pomocí Gitu.

5.2.1 Vytvoření projektu ve Visual Studio for Mac

Pro vytvoření projektu ve vývojovém prostředí Visual Studio for Mac se zvolí v hlavní nabídce File->New Solution a následně vyplní průvodce se třemi jednoduchými kroky. V prvním kroku 5.2 lze vybrat typ aplikace a případně specifický projekt pouze pro konkrétní platformu. V následujících dvou krocích se vyplní název balíčku a název aplikace5.3, respektive umístění zdrojových souborů 5.4.



Obrázek 5.2: Krok 1 – Výběr typu aplikace

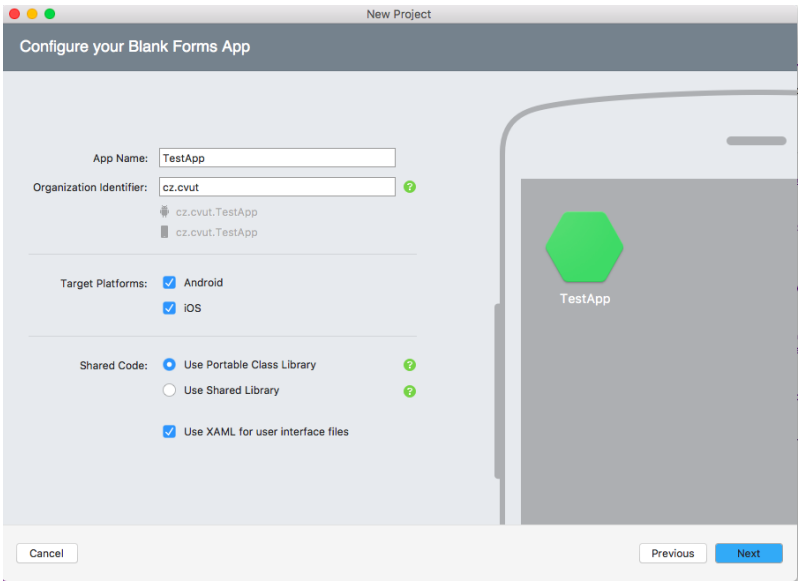
5.2.2 Spuštění a ladění aplikace

Pro spuštění aplikace Android je potřeba mít nainstalovaný Android SDK a příslušný emulátor. Simulátory pro macOS jsou součástí Xcode vývojového prostředí. V horní části vývojového prostředí 5.6 lze vybrat, který projekt má být spuštěn, v jakém režimu (Debug, Release) a na jakém zařízení. O simulátorech a emulátorech se diskutuje v kapitole 5.6.

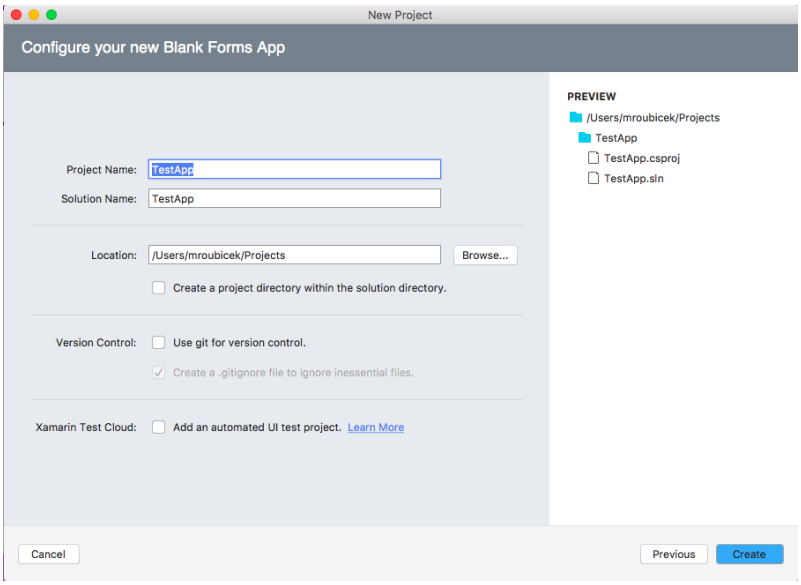
Graficky je prostředí rozloženo do několika dílčích podoken 5.7, které lze skrývat. Jednotlivá okna, která jsou užitečná při vývoji:

- **Textový editor** – Okno pro psaní zdrojového kódu. V záhlaví lze přepínat mezi otevřenými soubory. V konkrétním souboru lze navigovat nad metodami a třídami v daném souboru.
- **Breakpoints** – Přehled breakpointů v aplikaci, lze je dočasně vypínat/zapínat. Přidání se provádí kliknutím na okraj řádku v editovaném souboru, a řádek je následně označen červenou značkou.
- **Locals** – Přehled lokálních proměnných v daném kontextu. Proměnné lze upravovat za běhu aplikace dvojklikem do položky „Value“ v tomto okně

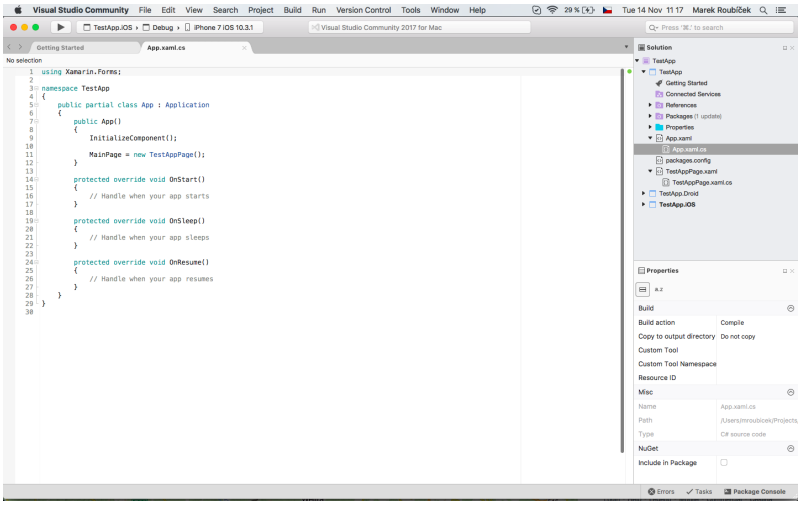
[6]Team foundation service - nástroj od Microsoftu pro správu, verzování projektů a sledování změn



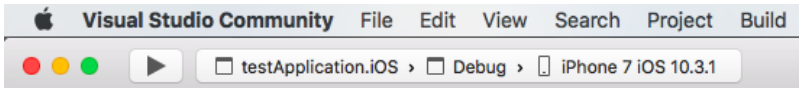
Obrázek 5.3: Krok 2 – Pojmenování aplikace a balíčku



Obrázek 5.4: Krok 3 – Pojmenování projektu a umístění zdrojových souborů

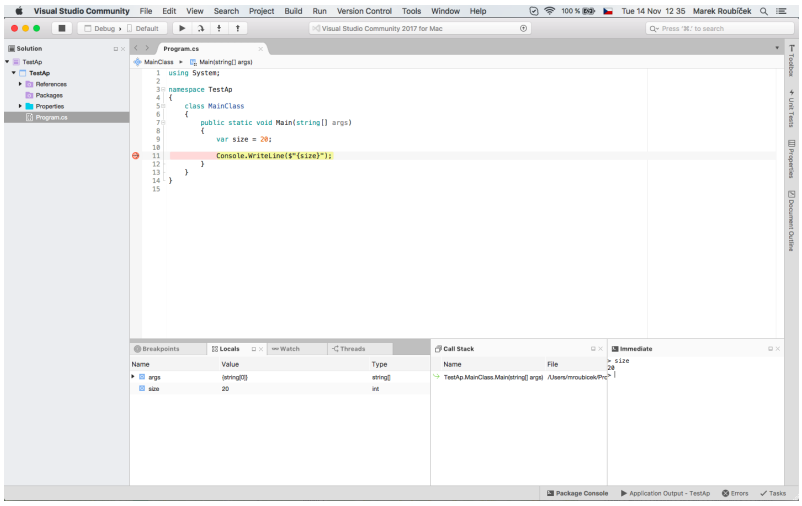


Obrázek 5.5: Náhled na projekt ve Visual Studio for Mac



Obrázek 5.6: Výběr aplikace ke spuštění v horní liště vývojového prostředí

- **Watch** – Seznam vlastních proměnných, případně vlastně vytvořených složitějších výrazů ke sledování
- **Call stack** – Přehled zásobníku volání programu, pomocí klikání v tomto okně se dá pohybovat po jednotlivých framech.
- **Immediate** – Okno pro volání okamžitě proveditelného kódu v daném kontextu při ladění, např. zjišťování hodnot proměnných, přiřazování apod.



Obrázek 5.7: Ladění ve vývojovém prostředí

5.3 Xcode

Xcode je vývojové prostředí pro macOS. Je potřeba ho mít nainstalován pro vývoj aplikací pro iOS aplikace. Nicméně díky můstku z Visual Studia není pro samotný vývoj potřeba, aby bylo zapnuté. Obsahuje vývojové nástroje včetně kompilátoru a simulátorů pro ladění aplikace. Pomocí Xcode se dále aplikace distribuuje na Itunes Connect, což je webový nástroj pro správu, testování a nasazování aplikací na AppStore. Této problematice se věnuje kapitola Nasazení aplikace na AppStore 7.2.

5.4 Workbooks

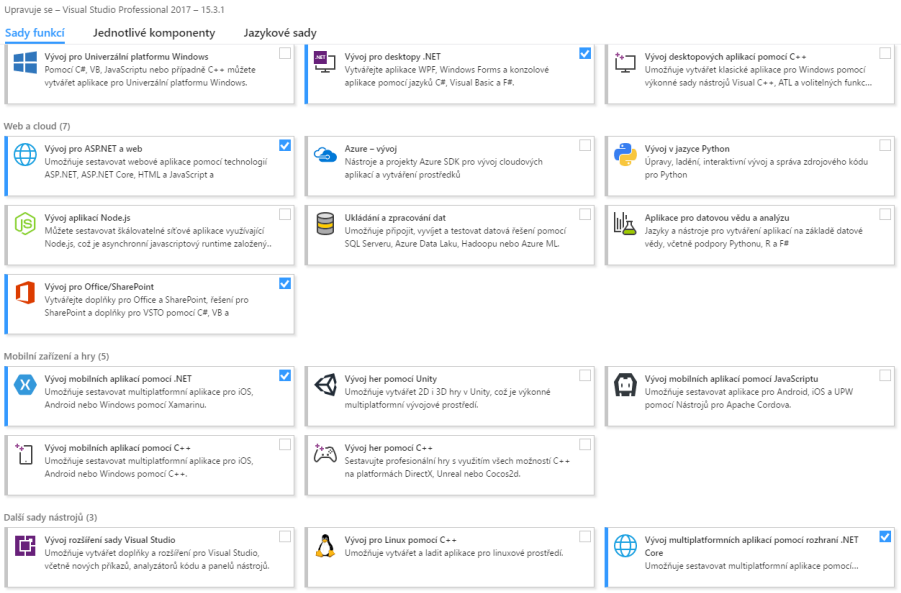
„Workbooks je interaktivní dokumentační nástroj pro objevování nespočtu oblastí, od samotných základů C# jazyka po podstatu computer science a pokročilá témata týkajících se vývoje mobilních aplikací.“ [38] Díky tomuto nástroji lze vytvářet samo vzdělávací dokumenty, které kombinují bloky reálného C# kódu s textovými popisky a vysvětleními. Vedle toho je možné C# kód okamžitě spouštět bez nutnosti kompilace nebo prozkoumávat jednotlivé vrstvy grafického prostředí v konkrétním příkladu. Nástroj má podobné prvky se systémem Wolfram Mathematica, včetně interaktivního vyhodnocování kódu.

5.5 Visual Studio 2017 Professional

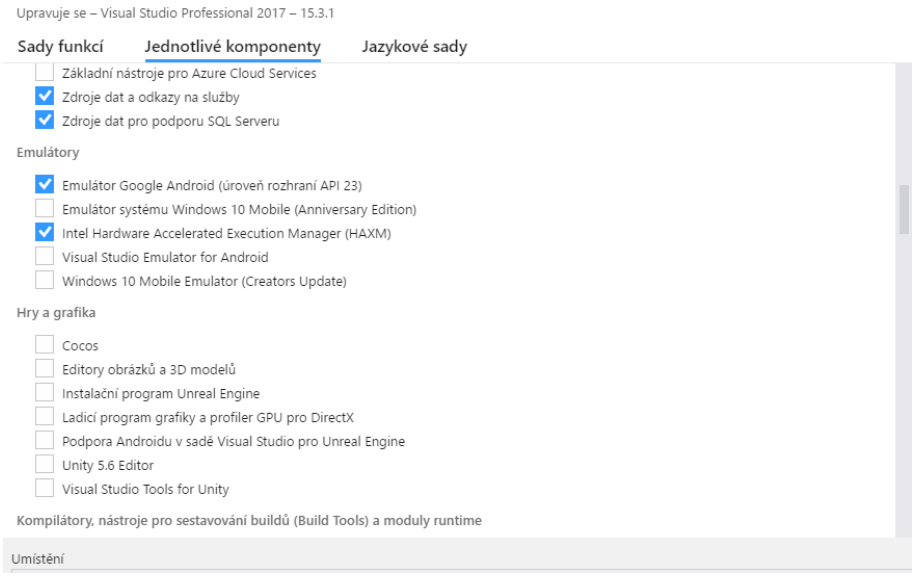
Pro tvorbu aplikací eRecept bylo použito vývojové prostředí Visual Studio 2017. Jedná se o nejnovější verzi hlavního vývojového prostředí jazyka C# pro operační systémy Windows. Podporuje tvorbu kódu v nejnovějším C# 7.

5.5.1 Instalace Visual Studio 2017

Vývojové prostředí se instaluje pomocí Visual Studio Installeru (obrázek 5.8, který umožňuje poskládat si vývojové prostředí dle vlastního výběru z jednotlivých modulů pro vývoj. Pro vytváření mobilních aplikací v Xamarinu je to modul „Vývoj mobilních aplikací pomocí .NET“ viz. obrázek. V záložce „Jednotlivé komponenty“ pak doporučuji pro úsporu místa (20 GB) odškrtnout instalaci defaultních emulátorů Android (obrázek 5.9, které se dají snadno doinstalovat i dodatečně. V původních verzích Visual Studia nešla dodatečně doinstalovat lokalizace, která se vybrala při instalaci defaultně dle nastavení operačního systému, což byla v mém případě čeština, pro běžně používanou angličtinu bylo potřeba IDE přeinstalovat. Tento problém byl již odstraněn.



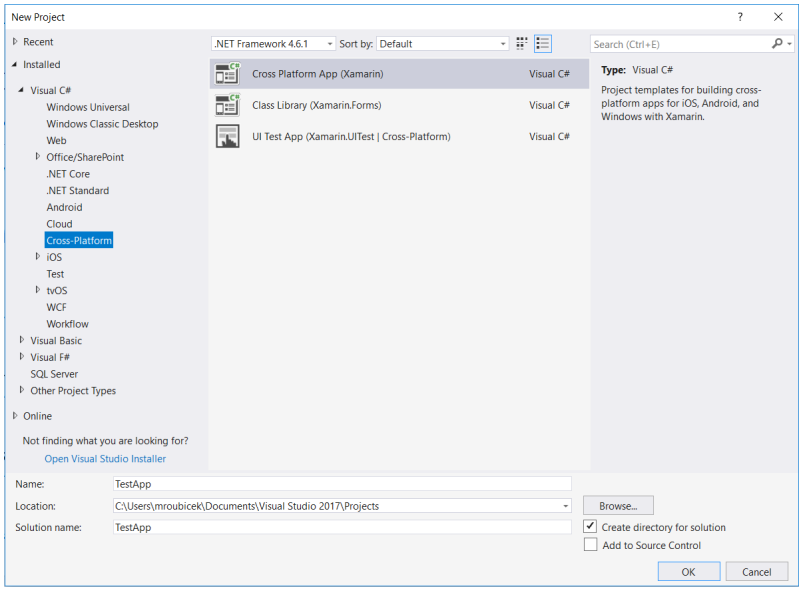
Obrázek 5.8: Instalace modulů Visual Studio 2017



Obrázek 5.9: Výběr komponent při instalaci Visual Studio 2017

5.5.2 Vytvoření projektu ve Visual Studio 2017

Vytvoření projektu se provádí pomocí nabídky **File->New->Project** a výběrem požadovaného typu projektu. Pro projekt používající Xamarin jsou pro vytvoření již dostupné šablony v záložce „Cross-Platform“ , které vytvoří dílčí projekty a soubory automaticky.



Obrázek 5.10: Vytvoření projektu Xamarin ve Visual Studio 2017

V druhém kroku viz. obrázek 5.10 lze vybrat, jaký projekt bude založen. Šablona se dá modifikovat 3 způsoby, čímž vzniká celkem 8 možností vytvořené aplikace:

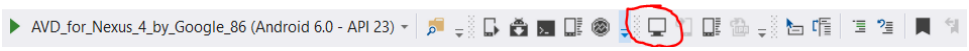
- **Typ aplikace** – Lze vybrat, zda se založí naprosto prázdná aplikace s výchozí jednoduchou stránkou nebo tzv. Master Detail, což je rozložení aplikace, ve kterém je jedna hlavní stránka a vyjíždějící menu z levého kraje obrazovky (viz. kapitola Pages).
- **UI Technologie** – Výběr technologie použité pro grafické prvky. Na výběr jsou možnosti
  - **Xamarin.Forms** – Používá knihovny Xamarin.Forms v cílových projektech
  - **Native** - Používá nativní knihovny (Zabalené do C# jazyka)
- **Strategie sdíleného kódu** – Výběr ze strategií pro vytvoření projektu se sdíleným kódem.
  - **Shared -** Projekt je při kompilaci brán jako součást kompilovaného projektu. Veškeré reference se vyhodnocují tak, jako kdyby byl kód v cílovém projektu.
  - **PCL** – Projekt se kompiluje samostatně – je možné přidávat reference na existující balíčky nebo naopak projekt znovupoužít v jiné aplikaci.

Aplikaci lze samozřejmě v průběhu vývoje měnit a rozšiřovat o nové doplňující projekty splňující vybranou strategii, což bylo uděláno i v případě finální aplikace v této práci. Na druhou stranu šablony pro vytvoření projektů slouží jako dobrý základ při vytvoření nové aplikace, neboť vytvoří všechny soubory zejména v platformně specifických projektech, které jsou potřeba pro konečné spuštění (například AndoidManifest.xml pro Android nebo info.plist pro iOS, což jsou soubory definující důležité informace aplikace jako číslo verze, identifikátor aplikace, oprávnění atd.). V současné době chybí v šablonách pro Xamarin třetí typ strategie pro sdílený kód, což je **.NET standard** 2.1 balíček, který je potřeba přidat ručně po vytvoření základní kostry projektu.

Projekty sdíleného kódu v aplikacích eRecept Pacient/Lékař byly zpočátku vytvořeny s čistě PCL strukturou, avšak v průběhu práce byly přidány další Shared projekty a v dalším postupu prací se přešlo z PCL na .NET Standard.

5.5.3 Připojení k Mac zařízení

Připojení k zařízení Mac je nutné ke kompilaci, spouštění i nasazování iOS a macOS aplikací z operačního systému Windows. Zároveň se jedná o nejsložitější krok celého vývoje. Pro spojení je potřeba, aby na obou propojovaných zařízeních byl nainstalován Xamarin SDK, který se nejjednodušeji nainstaluje společně s jednotlivými IDE. Ve Visual Studiu 2017 se připojení k Mac

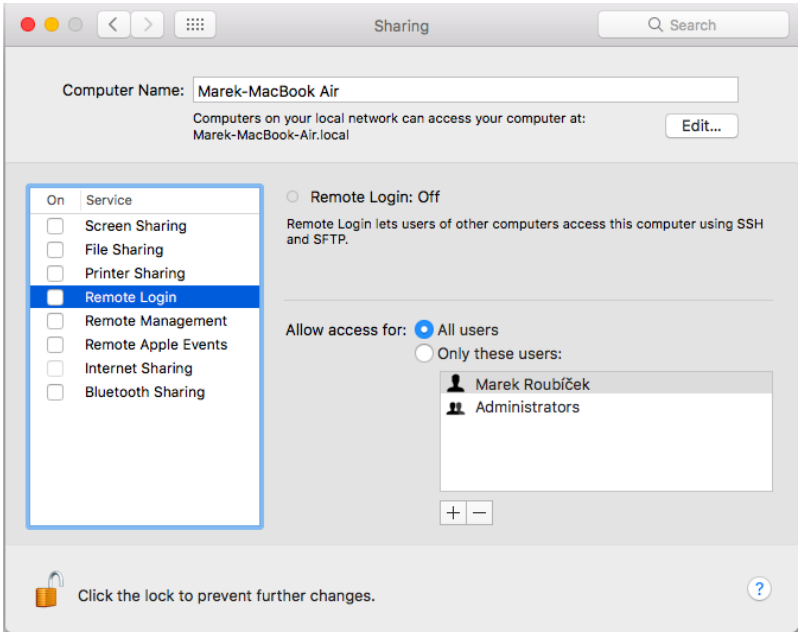


Obrázek 5.11: Ikona pro spuštění Xamarin Agenta

zařízení provádí v panelu nástrojů stisknutím ikonky monitoru spouštějící službu Xamarin Agent (viz. obrázek 5.11)

K tomu, aby mohl být cílový počítač nalezen je nejprve v nastavení Macu povolit vzdálené přihlašování a přístup. To se provádí v nastavení v sekci **Sharing** na konkrétním zařízení (viz. obrázek 5.12) a povolení možností **Remote Login** a **Remote Management** (obrázek 5.13).

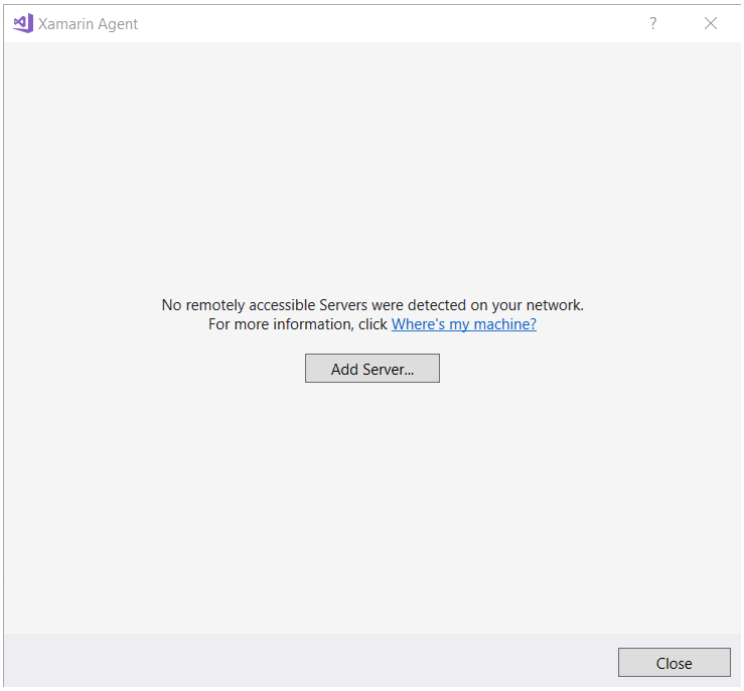
Obrázek 5.12: Nastavení sdílení na zařízení Mac



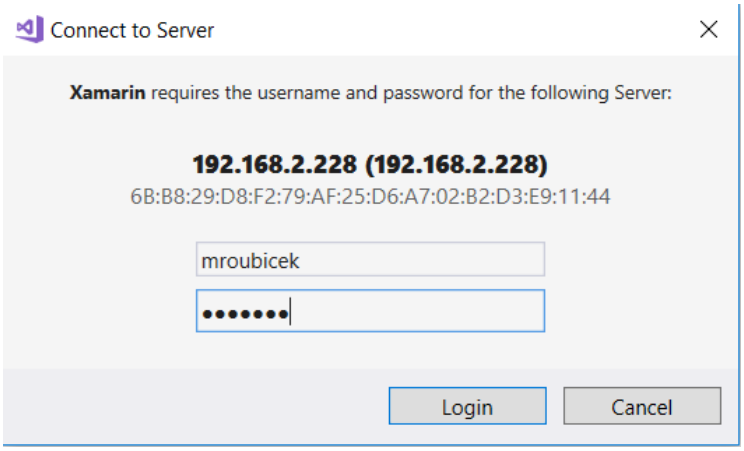
Obrázek 5.13: Povolení vzdáleného přihlašování a správy na zařízení Mac

Pokud je Mac zařízení přidáváno poprvé (obrázek 5.14) nebo není ve stejné síti jako zdrojový počítač, tak je potřeba přidat ručně IP adresu, na které se Mac nachází (obrázek 5.15). Následně je uživatel vyzván ke vzdálenému přihlášení (obrázek 5.16), a po potvrzení přístupu na straně Macu dojde ke spárování. Volba zařízení je uložena do lokální paměti, takže v budoucnu není potřeba zadávat přihlašovací údaje při každém spojení.

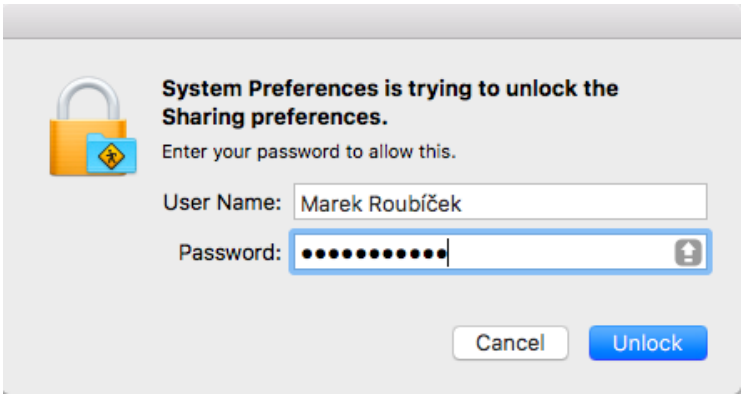
V případě, že se spojení povede, vyplní se monitor na ikoně v panelu nástrojů do zelena (viz. obrázek 5.17). Přes tuto ikonu lze opakovaně odpojovat a připojovat přidaná zařízení, při spuštění Visual Studia se systém pokouší automaticky o připojení, aby se ušetřilo zbytečné klikání. V



Obrázek 5.14: První připojení k zařízení Mac

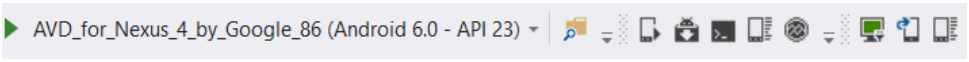


Obrázek 5.15: Připojení k Mac

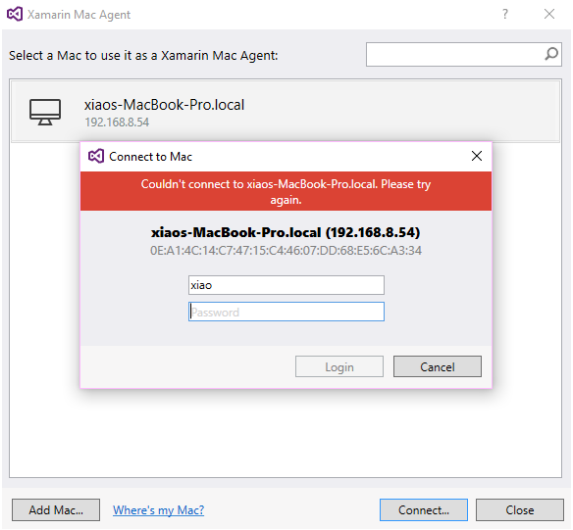


Obrázek 5.16: Výzva k potvrzení připojovaného zařízení

opačném případě se zobrazí v okně s přihlášením červená varovná hláška (viz. obrázek 5.18). Příčin může být hned několik, mezi nejčastější patří chybějící nebo špatná instalace Xamarin.IOS SDK na cílovém počítači nebo špatně zadané údaje. V případě, že problémy setrvávají, lze ještě vyzkoušet restart Visual Studia nebo celého počítače, jelikož propojování si ukládá do cache různé údaje a je nutné je vyčistit. Tento problém zmiňuji, protože jsem se s ním setkal zejména při počátku seznamování se s Xamarinem, jak se ale zdá, v novějších verzích frameworku byl tento problém odstraněn.



Obrázek 5.17: Indikátor úspěšného připojení



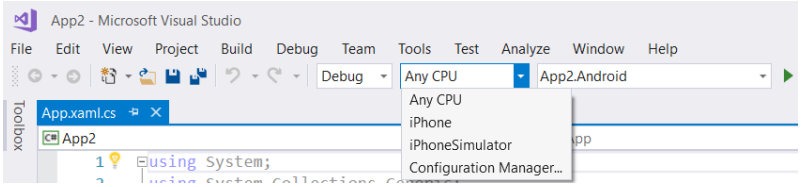
Obrázek 5.18: Nepovedené připojení k cílovému zařízení

5.5.4 Spuštění a ladění aplikace

Spuštění aplikace je velmi podobné postupu z podkapitoly Spuštění a ladění aplikace, tedy v horní části IDE je panel s možností výběru zařízení, na kterém se má aplikace spustit.

5.5.4.1 Spuštění aplikace pro Android

Pro platformu Android stačí vybrat reálné zařízení připojené pomocí USB, nebo některý z nainstalovaných emulátorů. Jediné, na co je potřeba dát si pozor, aby byl projekt sestaven pro požadovanou architekturu. Více o možných architekturách je napsáno v sekci Emulátory a simulátory. Pokud chceme, aby byla aplikace rovnou nahrána do zařízení je potřeba zaškrtnout volbu **Deploy** v Configuration manageru viz. obrázek 5.19.

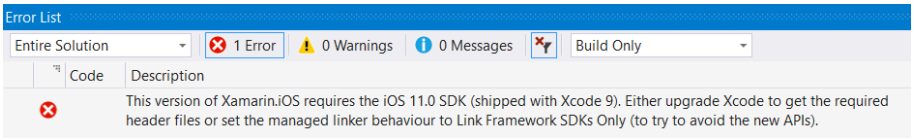


Obrázek 5.19: Configuration manager

5.5.4.2 Spuštění aplikace pro iOS

Pro spuštění aplikace na platformě iOS je potřeba mít spuštěné funkční spojení se zařízením Mac z předchozí kapitoly. Dále je potřeba zvolit správnou konfiguraci ve vlastnostech projektu s příponou „.iOS“ . Pro spouštění na simulátorech není na rozdíl od spouštění v reálném zařízení, aby byla

aplikace podepsána – podepisování aplikací certifikátem bude popsáno v kapitole 7.2.3.1. Na výběr jsou všechna připojená zařízení ke vzdálenému Macu a dále všechny simulátory, které se doinstalují s instalací Xcode na Macu. V případě, že nesedí vzájemně verze ať Xamarin SDK nebo Xcode a aktualizace Visual Studia, aplikace nepůjde spustit a vypíše se chybová hláška (viz. obrázek 5.20).

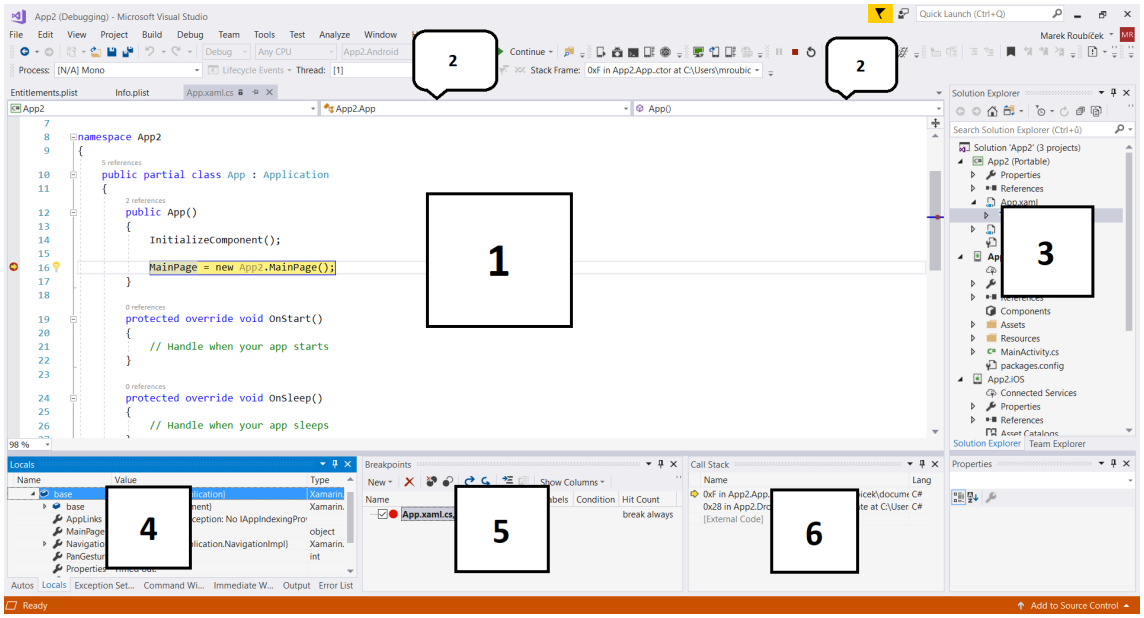


Obrázek 5.20: Chybová hláška špatná verze Xcode

5.5.4.3 Ladění aplikace

Ladění aplikace je již totožné pro obě vyvíjené platformy. Do textového editoru lze kliknutím na levý okraj řádku přidávat breakpointy. Při ladění mobilní aplikace není, na rozdíl od ladění jiných C# aplikací, možné vracet kontext přetažením myši z aktuálního řádku na předcházející nebo následující řádky. Navigovat programem můžeme pomocí tlačítek v horním panelu, nebo pomocí klávesových zkratek (Hodí se zejména **F10** pro krok vpřed, **F11** pro zanoření do metody, **Shift + F11** pro vynoření z aktuální metody a **F5** pro opětovné spuštění programu). Obrazovka je rozdělena do několika sekcí viz. obrázek 5.21:

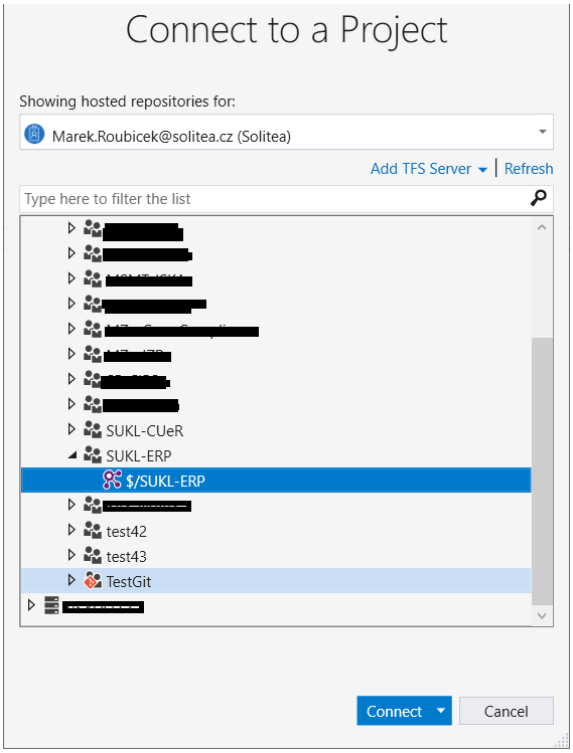
1. Textový editor
2. Navigace na třídu a metodu v otevřeném souboru
3. Stromová hierarchie projektu
4. Lokální proměnné
5. Breakpointy
6. Zásobník volání



Obrázek 5.21: Vývojové prostředí Visual Studio 2017 při ladění aplikace

5.5.5 Verzování pomocí TFS

Pro jednoduché verzování zdrojových kódů ve Visual Studiu 2017 lze využít TFS (Team foundation server). Pro připojení k serveru se v Team Exploreru zvolí možnost **Manage Connections**... **-> Connect to a Project**, zadají se přihlašovací údaje a adresa serveru, a poté se vybere příslušný projekt 5.22, ke kterému bude zdrojový kód na lokálním počítači přiřazen.



Obrázek 5.22: Připojení k projektu

Po každé změně (přejmenování/přidání/smazání souboru, změně v kódu) se soubor objeví v sekci **Pending changes** (viz. obrázek 5.23)v **Team Exploreru**. Změny, které lze tzv. „commit-nout[7]“ na server, jsou pod nadpisem **Included Changes**. Ty změny, které se mají nechat pouze na lokálním počítači (pod nadpisem **Excluded Changes**) se vyjmou kliknutím na akci **Exclude** nad vybranými soubory. Pro lepší vyhledávání v jednotlivých změnách lze ke každému souboru změn přidat komentář, který je vidět v historii.

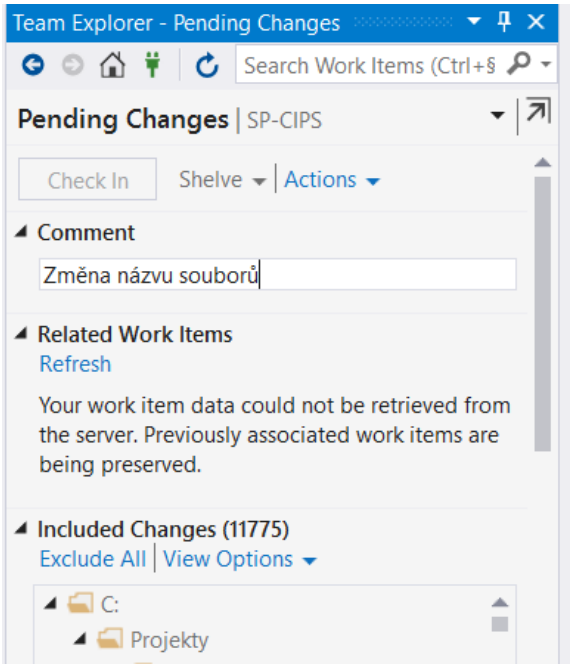
Pro opětovné stažení konkrétní verze, nebo stažení zdrojových kódů spolupracovníkem, se v Team Exploreru vybere sekce **Source Control Explorer**, která otevře okno s existujícími projekty na připojeném serveru. Nad vybranou verzí (obrázek 5.24 se vybere možnost **Get this version**, případně nad celým projektem **Get Latest Version.**

5.6 Emulátory a simulátory

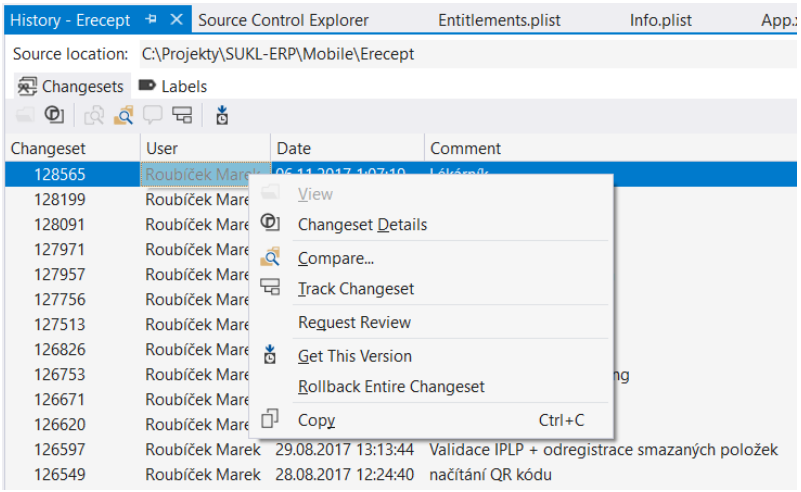
Aby bylo možné aplikace testovat a zkoušet bez fyzických zařízení, existuje možnost doinstalovat si na zařízení emulátor nebo simulátor. Jedná se v zásadě o program, který umožňuje instalovat, spouštět a zobrazovat aplikace tak, jako na cílové platformě. Zásadní rozdíl mezi emulátory a simulátory je ten, že emulátor se snaží simulovat všechny aspekty reálného zařízení – tedy software i hardware. Naproti tomu simulátor simuluje pouze softwarové vlastnosti cílové platformy[39], a nedokáže tedy 100% reflektovat změny a problémy hardwaru. Co se týká porovnání rychlosti, má ze své podstaty výkonovou výhodu simulátor, ale dnes již existují nástroje, které dokáží pomalou emulaci urychlit na srovnatelnou mez.

Pro testování v počátcích se mohou simulátory a emulátory hodit, jelikož představují jednoduchou cestu, jak otestovat aplikaci na celé škále zařízení, která nemusí mít vývojář fyzicky k dispozici. Zároveň odpadá problém při vývoji v týmu, jelikož vývojáři nemusí sedět v jedné kanceláři, aby

[7]Commit - potvrzení lokálních změn do TFS společného projektu



Obrázek 5.23: Sekce Pending Changes



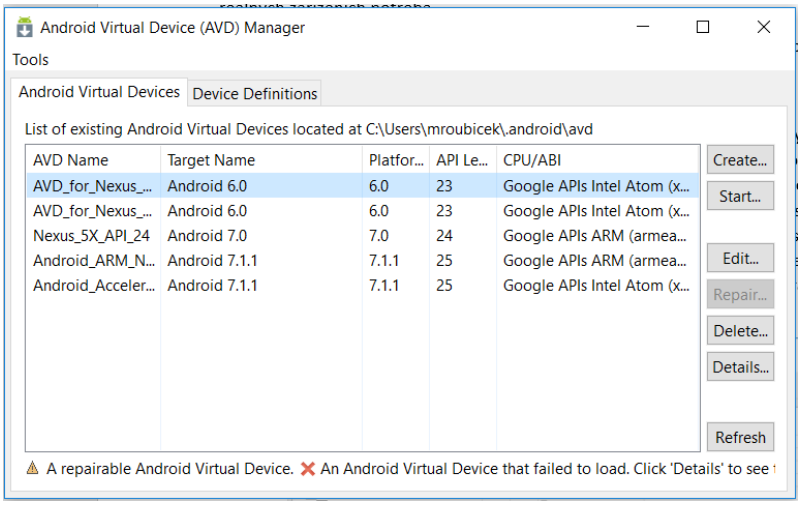
Obrázek 5.24: Výběr verze v Source control exploreru

mohli zařízení sdílet. Na druhou stranu pro nasazení do reálného provozu je otestování na několika reálných zařízeních potřeba.

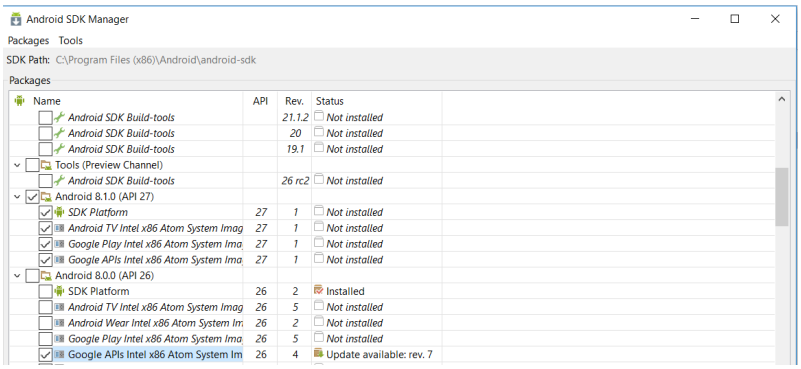
„Doporučený přístup k získání nejlepších výsledků vašeho testování je najít zdravý mix emulátorů, simulátorů a reálných zařízení“ [40]

5.6.1 Emulátory

Visual Studio 2017 nabízí pro testování Android aplikací emulátory od Microsoftu, nebo více odladěné emulátory přímo od Google, které mohou využívat hardwarovou akceleraci (**HAXM** viz. [41]) a výrazně tak zlepšit uživatelskou použitelnost, která se téměř blíží použití reálného zařízení. Instalace emulátoru se provádí přes **Android SDK Manager**, který se dá nainstalovat samostatně (obrázek 5.25), nebo společně s instalací Visual Studia. Pro vytvoření emulovaného prostředí je potřeba stáhnout tzv. image, což je obraz cílové platformy s konkrétní verzí Androidu (obrázek 5.26) – tento obraz se dá pak použít pro vytvoření podporovaného zařízení – tzv. **AVD** (např. telefonu, tabletu), existují i obrazy pro chytré televize a hodinky.



Obrázek 5.25: Instalace obrazu Androidu pro emulátor



Obrázek 5.26: Android virtual device manager

5.6.1.1 Nedostupné funkce

Přestože emulátory pokrývají jak softwarové, tak hardwarové vlastnosti cílového operačního systému (například informace o stavu baterie, poloha apod.), některé funkcionality nejsou běžnými emulátory dostupné. Pro emulátory používané při vývoji aplikací eReceptu to jsou tyto funkce:

- Bluetooth
- NFC
- SD card insert/eject
- Device-attached headphones
- USB

5.6.1.2 Obvyklé problémy s emulátory

Během vývoje na emulátorech se vyskytlo pár problémů, na které je třeba upozornit a popsat jejich řešení:

- **Problém s debuggerem**

  Občas se stane, že systém při kompilaci zamrzne, a kompilace pak není zcela dokončená. Když dojde k restartu Visual Studia, tak při opětovném spuštění aplikace na emulátoru vyskočí varovná hláška (viz. obrázek 5.27) o chybějících souborech (většinou knihovny Mono.Posix). Aplikace se sice spustí, ale debugger je odpojený, a nejde tedy aplikaci ladit zároveň se zdrojovými kódy. Řešení spočívá ve dvou krocích, buď jednom anebo kombinaci obou:

  1. Vyčištění projektu pomocí akce **Clean** a opětovné sestavení

2. Odstranění všech breakpointů, a následné přidání až po spojení s debuggerem

Obrázek 5.27: Varovná hláška debuggeru

- **Problémy při uspání PC**

  Pokud se počítač přepne do režimu spánku během spuštěného emulátoru (případně během debuggování aplikace), může na emulátoru nastat několik problémů spojené i s hardwarem zařízení, jelikož emulátor využívá systémové hodnoty a hardware.

  1. **Zamrznutí hodin na emulátoru**, které pak při opětovném probuzení počítače zůstanou na původní hodnotě – Řešením tohoto jevu může být buď ruční přenastavení hodin (ale dojde ke ztrátě synchronizace s hodinami systému), nebo restart celého emulátoru.
  2. **Ztráta ovladačů zvukové karty počítače** – Tento problém byl možná dost specifický při mém vývoji na zařízení **HP Elitebook 8440p**, nicméně jistě stojí za zmínku popsat jak situaci vyřešit. Problém se projeví tak, že notebook nemůže nalézt žádné zařízení pro přehrávání zvuku, a neumí si poradit ani pomocí systémové diagnostiky, ani pomocí restartu operačního systému. Jediné řešení je přepnout při bootování počítače do BIOS nebo UEFI a pomocí testu zvuku ovladače opět naběhnou.
  3. **Ztráta propojení s Visual Studiem** – Visual studio má díky službě Android Device Bridge(zkráceně ADB) přehled o připojených zařízeních a dostupných emulátorech. Taková zařízení pak nabízí jako volbu ke spuštění. Pokud je již emulátor spuštěn, tak se defaultně nepokouší spustit emulátor znovu, ale nahraje balíček automaticky. Někdy ale nastane problém, že se Visual Studio pokouší spustit emulátor nad stejným obrazem, jelikož došlo k odpojení od ADB. V tomto případě stačí vypnout původně spuštěný emulátor a pomocí spuštění aplikace nechat rozběhnout emulátor znovu.

5.6.2 Simulátory

Simulátory zařízení Apple jsou součástí Xcode vývojového prostředí, tudíž pokud se objeví nová verze operačního systému iOS, je třeba aktualizovat vývojové prostředí na Macu. Další rozdíl oproti emulátorům je, že na simulátor nelze nainstalovat aplikaci přímo z balíčku, ale pouze přes Xcode (Visual Studio 2017 toto řeší již zmíněným můstkem obou platforem).

Hardwarové vlastnosti zařízení nelze simulovat, to znamená, že funkce jako internetové připojení, klesající stav baterie a podobně, jdou otestovat pouze na reálném zařízení. Výhodou simulátorů je, že aplikace nemusí být při nahrání digitálně podepsaná, což razantně sníží dobu kompilace. Podrobný popis práce se simulátory lze najít přímo v dokumentaci Apple Developer[42].

5.6.3 Seznam použitých zařízení a simulátorů

Při vývoji byly využívány jek tablety, tak mobilní telefony a to jak v podobě fyzické, tak virtuální pro maximální možnou množinu rozdílných zařízení pro ověření správného fungování aplikace. Vzhledem k obrovskému množství různých zařízení není již dnes v lidských silách ověřit funkcionalitu na všech strojích v reálné době, proto existují různé komerční aplikace, které umožňují aplikaci testovat na propůjčeném hardware i vzdáleně – například aplikace HockeyApp. Pro tuto práci byla využita pouze některá zařízení:

Emulátory:

- Google APIs Intel Atom (x86) - Nexus 4
- Google APIs Intel Atom (x86) - Nexus 10
- Google APIs Intel Atom (x86_64)

Zařízení:

- Samsung Galaxy A3 Duos
- Samsung Galaxy S7

Simulátory

- Apple Iphone 5s
- Apple Iphone 7
- Apple Ipad

Zařízení:

- Apple IPhone 4S
- Apple Iphone 7
- Apple Ipad Air 2

Část II

Praktická část

Kapitola 6

Implementace aplikace

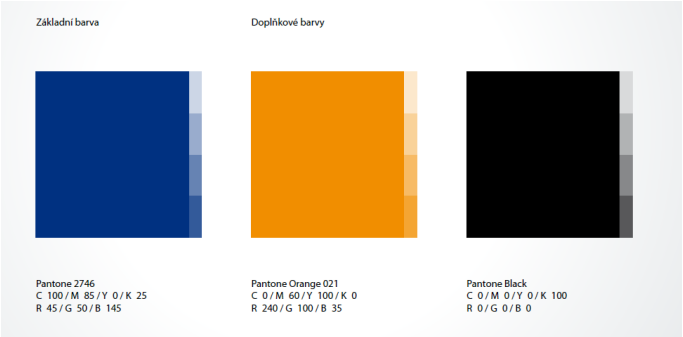
Součástí této práce byl samotný vývoj mobilních aplikací pomocí zvoleného nástroje pro vývoj multiplatformních aplikací. Při volbě technologie byly uplatněny zkušenosti společnosti Solitea business solutions s.r.o s vývojem aplikací v platormě .NET a zároveň s požadavky určené zadávací dokumentací od Státního ústavu pro kontrolu léčiv. Vzhledem ke skutečnosti, že výsledná aplikace je vlastnictvím Státního ústavu pro kontrolu léčiv, tato práce neobsahuje zdrojové kódy výsledného řešení, ale jen některé zajímavé ukázky a strukturu projektu – do zdrojových kódů lze v konkrétních případech nahlížet pouze po konzultaci s autorem této práce, proto některé ukázky může čtenář pochopit až v širším kontextu.

6.1 Požadavky

Na základě zadávací dokumentace byly definovány obecné požadavky na mobilní aplikace a zároveň specifické funkční a nefunkční požadavky pro pacienty a lékaře.

Obecné požadavky:

- Mobilní aplikace budou vytvořeny pro operační systémy iOS verze 9 a vyšší, Android 4.4 a vyšší.
- Uživatelské rozhraní mobilní aplikace, kontextové nápovědy a dokumentace bude minimálně v českém jazyce.
- Uživatelé budou mít v mobilní aplikaci k dispozici plnou kontextovou nápovědu a návod, jak s aplikací a jejími funkcionalitami pracovat.
- Barevné ladění obrazovek a formulářů systému e-Recept webové aplikace a mobilní aplikace bude v souladu s Design manuálem SÚKL (obrázek barevné schema aplikací SÚKL 6.1).



Obrázek 6.1: Barevné schema aplikací SÚKL dle Design manuálu

6.1.1 Aplikace Pacient

Mobilní aplikace pro pacienty bude sloužit všem potenciálním pacientům, kteří se rozhodnou přijímat elektronické recepty pomocí mobilní aplikace bez dalších papírových dokladů a průvodek, ale zároveň i jako přehled jejich již vyzvednutých léků za uplynulý rok. Dle zadávací dokumentace byly definovány tyto požadavky:

Nefunkční požavavky:

Aplikace bude

- používat set webových služeb pro komunikaci s CÚeR pro všechny akce.
- používat uživatelské jméno a heslo nastavené jen pro mobilní aplikaci ztotožněných pacientů pro autentikaci každého http požadavku (nastavováno přes webovou aplikaci).
- uvádět v každém požadavku IMEI mobilního zařízení (registrováno přes webovou aplikaci).
- používat pro HTTPS komunikaci s webovými službami pouze serverový certifikát.
- Mobilní aplikace nebude plně nahrazovat SW řešení třetích stran, a tedy nemusí umožňovat nadstavbové funkcionality, které slouží především pro snadnější práci uživatele.

Funkční požadavky:

- Systém pacientovi poskytne lékový záznam pacienta (ELZ).
- Systém umožní pacientovi získat výpis o jemu předepsaných a na předepsaný recept vydaných léčivých přípravků (LP).
- Systém umožní zobrazit seznam všech receptů vystavených na daného pacienta.
- Systém umožní filtrování seznamu podle zadaných filtrovacích podmínek.
- Systém umožní vygenerování průvodky receptu do PDF.
- Autentizace proběhne na základě uživatelského jména, hesla a IMEI mobilního zařízení.

Na základě požadavků byly upřesněny některé funkcionality, které bude aplikace poskytovat:

Přihlašovací obrazovka – Obsahuje pole pro zadání uživatelského jména a hesla. Autentizace bude prováděna přes webovou službu - pomocí přístupových údajů pro mobilní aplikaci nastavených ve webové aplikaci. Pokud uživatel nebude mít dosud nastavené přístupové údaje pro mobilní aplikaci tak bude vybídnut k přihlášení do webové aplikace, povolení mobilního přístupu a registraci zobrazeného IMEI.

Seznam receptů – Obrazovka bude obsahovat seznam nevyzvednutých receptů ve stručné podobě s možností rozkliknutí detailu jednotlivých položek. Zároveň bude možné pomocí kontextového filtru přes položky vyhledávat v předepsaných receptech. Pacient uvidí a bude moci vyhledávat pouze svoje eRecepty. Vyhledávání bude možné vyhledávat i podle jeho identifikátoru a to buď ručně zadaném nebo pomocí načtení čárového nebo QR kódu fotoaparátem mobilního zařízení.

Detail receptu – Detail bude obsahovat informace o léčivých přípravcích, s možností prokliku na webu SÚKLu s kóodem daného léčiva. V detailu eRp bude na rozklik zobrazeny i jeho výdeje v lékárnách s možností náhledu na detaily jednotlivých výdejů. Na detailu receptu bude možné stáhnout průvodku eReceptu v PDF formátu.

6.1.2 Aplikace Lékař

Aplikace pro lékaře bude sloužit pro předepisování eReceptů certifikovaným lékařům, kteří nebudou chtít pro vydávání eReceptů využívat tlustého klienta[8] na svém počítači v ordinaci (ať už

[8]Systém, kde maximum zpracování dat probíhá na vlastním počítači a server slouží jen pro archivaci a komunikaci

z důvodu nedostupnosti síťového připojení nebo z důvodu práce mimo ordinaci). Pro používání aplikací bude nutné mít zřízen uživatelský účet včetně všech ověřených certifikátů dle předpisů SÚKLu.

Zadávací dokumentace definuje pro aplikaci lékaře tyto požadavky:

Nefunkční požadavky:

Aplikace bude

- Používat obvyklý set služeb pro uživatele typu lékař.
- používat uživatelský účet lékaře v Externích identitách pro autentikaci každého http požadavku.
- používat pro komunikaci klientský HTTPS přístupový certifikát vydaný systémem Externích identit uložený s privátním klíčem v mobilním zařízení v podobě souboru.
- používat pro elektronický podpis předepisovaných eRp digitální certifikát uložený s privátním klíčem v mobilním zařízení nebo na externím úložišti certifikátů.
- Mobilní aplikace pro lékaře bude fungovat obdobně jako informační systémy pro lékaře poskytované třetími stranami. Rozdíl bude jen v tom, na jakém zařízení běží a její funkčnost bude oproti plné funkčnosti obvyklých SW pro lékaře omezena jen na předepisování receptů a náhled na ně.

Funkční požadavky:

- Systém umožní vystavení receptu pomocí mobilní aplikace.
- Systém umožní v mobilní aplikaci vytisknout průvodku (PDF).
- Systém umožní zobrazit seznam všech receptů vydaných daným lékařem.
- Systém umožní filtrování seznamu podle zadaných filtrovacích podmínek.
- Autentizace proběhne na základě uživatelského jména, hesla a klientského certifikátu.

Na základě požadavků byly upřesněny některé funkcionality, které bude aplikace poskytovat:

Přihlašovací obrazovka – Obsahuje pole pro zadání uživatelského jména, hesla a čísla pracoviště daného uživatele. Autentizace bude prováděna přes webovou službu - pomocí přístupových údajů pro mobilní aplikaci a nainstalovaného klientského certifikátu.

Vystavení receptu – Výchozí obrazovka po přihlášení umožní založení nového receptu – tedy vyplnění údajů pacienta, předepisovaných léčivých přípravků včetně množství, údajů o platnosti eReceptu a způsobu notifikace pacienta o novém receptu a dalších položek. Aplikace umožní vystavit pouze recept se správně vyplněnými položkami, včetně kontroly vyplnění povinných hodnot. Po úspěšném založení je možné zobrazit detail receptu a vytisknout průvodku.

Na recept bude možné zadat maximálně 2 položky (konfigurace systému) vzhledem k aktuální úpravě legislativy, kdy na papírové recepty lze uvádět pouze dva léčivé přípravky.

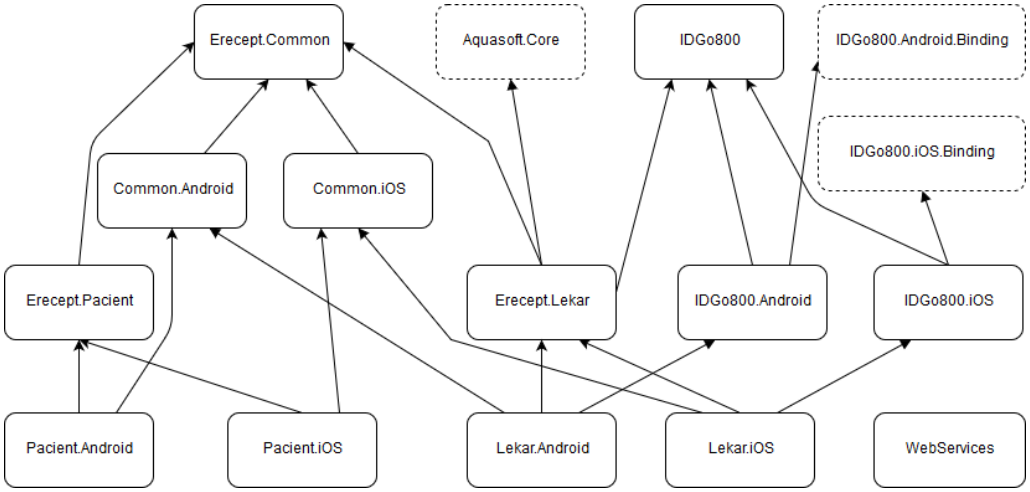
Seznam receptů – Obrazovka bude obsahovat seznam receptů ve stručné podobě s možností rozkliknutí detailu jednotlivých položek. Zároveň bude možné pomocí kontextového filtru přes položky vyhledávat v předepsaných receptech. Při vyhledání receptů podle identifikátoru receptu může uživatel zobrazit jakýkoliv recept, i který nevystavil. V případě, že není ve filtru zadaný identifikátor receptu, zobrazí se uživateli jen ty recepty, které sám vystavil.

Detail receptu – Detail bude obsahovat informace o léčivých přípravcích, s možností prokliku na webu SÚKLu s kódem daného léčiva. V detailu eRp bude na rozklik zobrazeny i jeho výdeje v lékárnách s možností náhledu na detaily jednotlivých výdejů. Na detailu receptu bude možné stáhnout průvodku eReceptu v PDF formátu. Na rozdíl od detailu receptu v aplikaci pro pacienty bude u lékařů možné prohlížet si údaje o pacientovi.

6.2 Projekt eRecept

Struktura solution jako je vidět na obrázku 6.2 je tvořena hierarchickými závislostmi mezi jednotlivými projekty (Znázorněno šipkami mezi jednotlivými projekty) tak, aby byla co největší množina funkcí sdílena mezi jednotlivé aplikace, a zároveň, aby každý projekt obsahoval jen potřebnou funkčnost. V následujících podkapitolách se tato práce zaměřuje na jednotlivé projekty s vysvětlením jejich účelu a zmínění některých klíčových prvků každého projektu. Projekty zobrazené přerušovanou čárou byly též vytvořeny Solitea Business Solutions s.r.o, nicméně v této práci bude jejich význam vysvětlen pouze okrajově. Veškerá další závislosti na knihovnách a doplňcích třetích stran nejsou v této práci z důvodu bezpečnosti explicitně zmíněny.

Struktura je takto navržena, jelikož není členěna pouze na podle platforem (iOS a Android), ale zároveň je členěna podle typu aplikace (Lékař, Pacient). Tudíž existují celkem 4 varianty výsledné konfigurace. Díky této struktuře bylo v průběhu práce možné relativně snadno (v rámci dní) doimplementovat další aplikaci pro lékárny opět na obě zmíněné platformy.



Obrázek 6.2: Graf závislosti jednotlivých projektů v Solution eRecept

Aplikace byla vyvíjena pomocí frameworku **Xamarin.Forms**, tudíž splňuje architekturu MVVM (Model – ViewModel – View) s využitím základních ovládacích prvků a konceptů (Custom renderers, Dependency Injection, XAML atd.). Do aplikace byly navíc importovány knihovny třetích stran pro práci s čárovými kódy, databází, dialogy atd.

6.2.1 Názvy tříd, souborů a projektů

Projekt byl vytvořený v angličtině, proto všechny třídy, metody, komentáře, soubory a projekty mají anglické názvy s výjimkou dvou pojmů – „**Pacient**“ a „**Lekar**“ , pro jednodušší vyhledávání při vývoji.

6.2.2 Návrh grafického rozhraní

Aplikace psané ve frameworku **Xamarin.Forms** mají definovánu množinu možností grafického rozhraní tak, aby splňovali podmínku nativního vzhledu pro konkrétní platformu a zároveň bylo jejich ovládání intuitivní. Vzhledem k požadavkům na aplikaci bylo vhodné, aby aplikace:

- obsahovala menu – kvůli přechodu mezi jednotlivými obrazovkami
- pro složitější obrazovky umožnila členění pomocí záložek
- umožnila vrstvení obrazovek pro zpětné přechody (Například z detailu zpět na seznam)
- zobrazovala data kompaktně a přehledně
- odpovídala graficky webové aplikaci

Výše zmíněné podmínky vedli k následující hierarchii:

- Základní ovládací prvky jako tlačítko, text, plné pozadí budou vyvedeny v barvách dle grafického manuálu SÚKL
  - Nastavení stylů ovládacích prvků bude dle uvedené barevné kombinace
- Aplikace bude mít skrývací menu – standardně z levé části obrazovky zobrazitelné pomocí gesta posuvu z okraje ke středu displeje
  - Hlavní obrazovka, která bude postupně obsahovat jednotlivé hierarchocké vrstvy bude **MasterDetailPage**
  - V property Master dané Page bude samostatná stránka s menu
- Aplikace umožní vrstvení obrazovek
  - V property Detail hlavní Page bude **NavigationPage**
  - V nastavení platformy Android bude povolen toolbar (pro možnost přidávání ikonek na horní lištu v aplikaci) – platforma iOS toto nastavení obsahuje implicitně
- Složitější obrazovky budou členěny pomocí záložek
  - Stránky vyžadující složitější členění budou obsahovat jednotlivé záložky pomocí nadřazené struktury **TabbedPage**
- Odpovídala graficky webové aplikaci
  - Obrázek na pozadí obsahových stránek bude totožný s hlavním pozadím webové aplikace
  - Rozložení grafických prvků (zadávacích nebo zobrazovacích) bude v podobném nebo shodném uspořádání, a prvky budou ve stejném pořadí

6.2.3 Common

Projekty, které zahrnují společnou funkcionalitu pro všechny vyvinuté i potenciální aplikace patří du skupiny Common, které je vyhrazena vlastní složka v Solution. V těchto projektech jsou obecné koncepty mobilní aplikace – například grafické rozložení obrazovek a přechod mezi nimi nebo definice vlastních ovládacích prvků v jednotlivých platformách, viz. kapitola 3.3.

6.2.3.1 Erecept.Common

Výchozím projektem, který je společný napříč platformami i napříč aplikacemi je **.NET Standard** projekt **Erecept.Common**, který sdružuje většinu společné funkcionality. Aplikace je napsána ve frameworku **Xamarin.Forms**, tudíž tento projekt obsahuje koncepty Dependency Injection a ovládací prvky z této knihovny (Page, View, Cell, Layout atd.). Projekt je členěn do logických bloků podle svého účelu – v tomto projektu najdeme:

- **Data** - Třídy konfigurace (adresy endpointů webových služeb, konstanty další nastavení pro jednotlivé aplikace) a dále třídy pro práci s lokální databází. Lokální databáze se využívá pro:
- Ukládání stavu aplikace (Zobrazování obrazovky při prvním spuštění)
- **Stažené číselníky** – Některé položky v aplikaci (Diagnózy, Léčivé přípravky, Suroviny pro výrobu léčiv nebo Odbornosti) nejsou konstantní a mohou se v průběhu času měnit. Seznamy s číselníkovými hodnotami jsou umístěny na speciální serverové URL, kde se nachází i obecná služba s čísly verzí všech číselníků. Při přihlášení se aplikace dotazuje na aktuální verze a v případě aktualizací je uživatel informován o možnosti stažení nových hodnot.
- **Dependency** – Tento logický blok obsahuje:
  - rozhraní pro Dependency Injection (práce s přihlašovacími údaji skrz interní úložiště jednotlivých platforem, práce se soubory, komunikace se serverem pro volání webových služeb a načítání dat číselníků atd.)

– Třídu **DependencyFactory** návrhového vzoru Factory, která se používá pro inicializaci ovládacích prvků (především Page) a ViewModelů, které jsou různé podle toho, zda je aplikace pro . V kódu, kde jsou potřeba instance těchto tříd je volána metoda ve Factory, o konkrétní implementaci se pak stará potomek třídy DependencyFactory umístěn v každé aplikaci. Ukázka vytvoření rozdílných instancí přihlašovacích obrazovek skrze tuto třídu je patrná ze zdrojového kódu:

Common projekt
DependencyFactory

```
/// <summary>
    /// Factory for creating application specific (Lekar/Pacient)
        instances
    /// </summary>
    public class DependencyFactory
    {
        #region Properties

        private static DependencyFactory factory;
        public static DependencyFactory Factory
        {
            get
            {
                return factory;
            }
            private set
            {
                factory = value;
            }
        }
        /// <summary>
        /// Initialization of factory instance
        /// </summary>
        /// <param name="factory">Application specific implementation of
            factory</param>
        protected static void Init(DependencyFactory factory)
        {
            Factory = factory;
        }
        /// <summary>
        /// Creates <see cref="LoginPage"/> instance
        /// </summary>
        /// <param name="root"><see cref="LoginPage"/></param>
        /// <param name="detailPage"><see cref="NavigationPage"/></param>
        /// <returns><see cref="LoginPage"/> instance</returns>
        public virtual LoginPage CreateLoginPage(NavigationPage
            detailPage)
        {
            return new LoginPage(detailPage);
        }

  /* ZBYTEK ZDROJOVEHO KODU */
}
```

Třída DependencyFactory obsahuje statickou referenci na instanci téže třídy. Konkrétní instance je pak iniciována v projektu konkrétní aplikace. Zároveň existuje instanční virtuální metoda **CreateLoginPage**, která vrací instanci LoginPage, což je přihlašovací obrazovka. Pokud dědičná třída této Factory nepřepíše tuto metodu, je vytvořena základní alternativa LoginPage.

V **Common** projektu pak stačí zavolat metodu **CreateLoginPage** nad obecnou Factory třídou, a je vytvořena instance podle dané aplikace:

```
DependencyFactory.Factory.CreateLoginPage(detailPage);
```

Lekar projekt
App

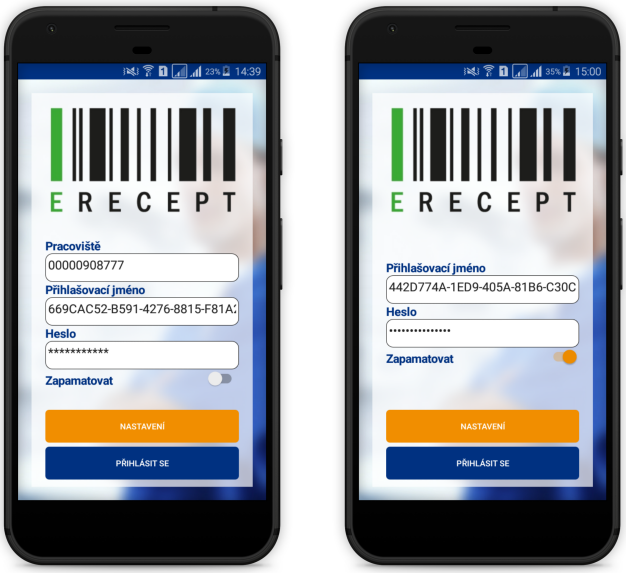
```
LekarDependencyFactory.Initialize();
```

Řádek ve třídě App, která je vstupním bodem aplikace pro lékaře, jenž provádí inicializaci instance dědičné Factory (metoda **Initialize** se nachází pod touto sekcí).

LekarDependencyFactory

```
        /// <summary>
        /// Factory for creating application specific instances for Lekar
            Application
        /// </summary>
        public class LekarDependencyFactory : DependencyFactory
        {
            #region Properties

            #endregion

            #region Static methods

            /// <summary>
            /// Initialize instance of app specific factory to ran
                application
            /// </summary>
            public static void Initialize()
            {
                Init(new LekarDependencyFactory());
                Configuration.AppType = ApplicationType.LEKAR;
    }
            /// <summary>
            /// Creates <see cref="LekarLoginPage"/> instance
            /// </summary>
            /// <param name="detailPage"><see cref="LoginViewModel"/></param>
            /// <returns><see cref="LekarLoginPage"/> instance</returns>
            public override LoginPage CreateLoginPage(NavigationPage
                detailPage)
            {
                return new LekarLoginPage(detailPage);
    }

    /* ZBYTEK ZDROJOVEHO KODU */
}
```

Poděděná třída LekarDependencyFactory odvozená od třídy DependencyFactory obsahuje metodu **Initialilze**, která provede naplnění instance do property Factory. Zároveň tato třída obsahuje přepsanou metodu **CreateLoginPage**, která vytvoří explicitní variantu LoginPage (Přihlašovací obrazovka obohacená o možnost zadat pracoviště a nastavit klientský certifikát přihlašujícího se lékaře 6.3).



(a) (b)

Obrázek 6.3: Rozdílná grafická podoba přihlašovací obrazovky aplikací Lékař(a) a Pacient(b) na platformě Android

- **Models** – Obsahuje entity pro práci ve ViewModelech a entity číselníků, které jsou uložené v databázi. Entity slouží pro snadnější Data Binding ve View.
- **Resources** – Soubory pro lokalizaci aplikace. Všechny textové řetězce jsou uloženy v RESX souboru, a v aplikaci je na ně vytvořen pouze odkaz. Aplikace jsou momentálně nastavené pro jazyk Angličtina a Čeština. Volba aplikace se provádí automaticky na základě nastavení jazyka operačního systému.
- **Services** – Pomocné statické třídy s jednoduchou funkcionalitou (Parsování URL, práce s dialogy v aplikaci, pomocná třída pro převod odpovědi webové služby na viewModely atd.)
- **UI** – Třídy v XAML nebo CS formátu pro grafické rozhraní aplikací
- **Controls** – Obsahuje vlastní ovládací prvky, poděděné od obecných konceptů frameworku Xamarin.Forms.
- **Pages** – Třídy jednotlivých Page, které jsou použity jako View
- **Templates** – Třídy, které slouží jako Data Template pro seznamové ovládací prvky (ListView) v jednotlivých Page
- **ViewModels** – ViewModel třídy pro jednotlivé obrazovky(View) v podobě Page v UI. Oddělují vlastní logiku a zabalují model. Vytvořená třída je pak Binding contextem pro Data Binding dané stránky.
- **BaseApplication** – Třída Application, která je společným vstupním bodem do aplikace. Definují se v ní mimo jiné obecné grafické styly ovládacích prvků. Projekty jednotlivých aplikací mají ve své struktuře potomky této třídy, přes které je výsledná aplikace spouštěna.

6.2.3.2 Common.Android

Common.Android je projekt, který obsahuje soubory a třídy, které jsou společné pro všechny aplikace eRecept na platformě Android. Od této třídy následně dědí koncové projekty Androidu pro jednotlivé aplikace (Lékař, Pacient). Tento projekt obsahuje základní soubory a třídy pro Xamarin.Android (viz. kapitola 3.3.2), včetně společných obrázků aplikace. Dále je v tomto projektu implementace Custom rendererů pro jednotlivé ovládací prvky z projektu **Erecept.Common**.

6.2.3.3 Common.iOS

Obdobně jako v případě Common.Android projektu obsahuje projekt Common.iOS základní soubory a třídy pro platformu iOS dle kapitoly 3.3.3. Dále pak implementace Custom rendererů pro ovládací prvky ze společného projektu.

6.2.3.4 Common.Shared

Projekt Common.Shared obsahuje třídy, které musí být dle struktury Xamarin projektů součástí platformních projektů, ale svým zdrojovým kódem jsou pro obě platformy naprosto totožné – z toho důvodu není potřeba vytvářet duplicitní zdrojové kódy a případné drobné rozdíly lze vyřešit pomocí direktiv kompilátoru. V tomto projektu jsou tedy implementace obecných rozhraní dle konceptu Dependency Injection.

6.2.3.5 Lekar, Pacient, Lekarnik

Konkrétní aplikace jsou zahrnuty v samostatných skupinách Lekar a Pacient. Svojí strukturou se neliší od obecných Common projektů – obsahují pouze nové obrazovky, nová rozhraní a jejich implementace, modely a další třídy v přidružených projektech, a nebo rozšiřující třídy (viz. příklad LekarLoginPage), které jsou v obecné struktuře vytvářeni pomocí návrhového vzoru Factory.

Struktura projektů je tedy opět totožná, pro úplnost pouze v seznamovém výpisu:

- Lekar
  - Erecept.Lekar
  - Lekar.Android
  - Lekar.iOS
  - Lekar.Shared
- Pacient
  - Erecept.Pacient
  - Pacient.Android
  - Pacient.iOS
  - Pacient.Shared

6.2.3.6 Lekarnik

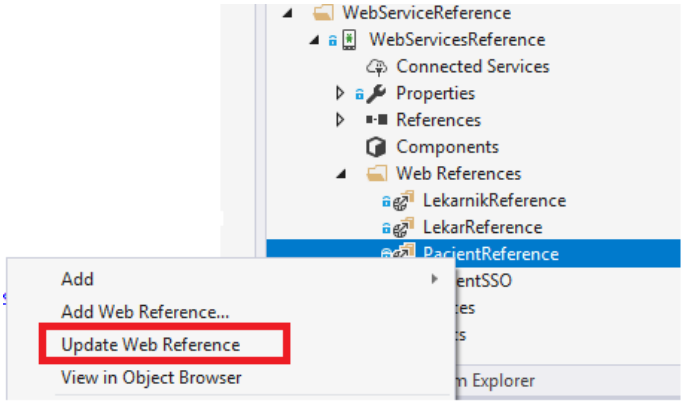
Během vývoje aplikací, kterými se zabývá i tato práce vznikl ze strany Státního ústavu pro kontrolu léčiv požadavek na další aplikaci pro lékárny, kterými budou moci lékárníci bez internetového připojení nebo mimo své pracoviště načtené eRecepty konkrétního pacienta zablokovat pro výdej ve své lékárně. V této práci se této aplikaci nevěnuji, nicméně struktura projektů je naprosto shodná jako v předchozích dvou případech (díky správně zvolené strategii sdílení kódu byla aplikace pro lékárníky s velmi malou vlastní nadstavbovou funkcionalitou vyvinuta během jednoho týdne)

Struktura projektů lékárníka vypadá následovně:

- Lekarnik
  - Erecept.Lekarnik
  - Lekarnik.Android
  - Lekarnik.iOS
  - Lekarnik.Shared

6.2.3.7 WebServicesReference

Samostatným projektem, který stojí úplně stranou od zmíněných projektů je projekt WebServicesReference, který slouží pouze pro snadnější správu vytváření tříd webových služeb z WSDL souboru. Jelikož Visual Studio umožňuje vygenerovat automaticky třídy (Potomek třídy **SoapHttpProtocol**), které jsou schopny komunikovat s webovými službami skrze SOAP protokol, v tomto projektu jsou tyto reference uchovány a jednoduchým kliknutím lze aktualizovat vygenerovaný soubor (viz. 6.4. Bohužel reference (ač společná pro obě platformy) může být pouze na platformním projektu (Xamarin.Android resp. Xamarin.iOS), proto tento projekt sdružuje všechny WSDL služby a následně jsou vkládány jedna ku jedné do Shared projektů jednotlivých apikací.



Obrázek 6.4: Aktualizace generované třídy na základě WSDL

6.2.4 Ostatní projekty

Součástí solution eRecept jsou i další projekty, které byly též vyvinuty společností Solitea Business Solutions s.r.o. Jejich struktura a fungování není obsahem této práce, neboť by vydaly na samostatnou kapitolu, proto jsou zde jen pro úplnost uvedeny včetně jejich účelu:

IDGo800

Na základě požadavku na aplikaci pro lékaře, který měl za cíl implementaci podepisování vytvořených eReceptů pomocí certifikátu na externím úložišti (tzv. SmardCard Token), projekty sdružené do složky IDGo800 obsahují logiku pro práci s externím úložištěm. V tomto projektu bylo potřeba vytvořit binding na API v nativních jazycích (Java a Objective-C) poskytnuté dodavatelem externích úložišt Gemalto, které umožňuje komunikace se čtečkou pomocí technologie BLE (Bluetooth Low Energy). Jednotlivá API jsou rozdílná, proto projekt obsahuje i nadstavbové rozhraní pro zahrnutí do .NET.

Aquasoft.Core

Tato složka obsahuje projekty, které byly vytvořeny jako framework pro vývoj projektů v .NET napříš technologiemi (MVC, Xamarin, WPF atd.). Obsahují obecné funkcionality jako např. logování nebo komunikaci skrze webové služby a kryptografické funkce.

6.2.5 Funkčnosti aplikace

Vzorový průchod aplikací pro pacienta na platformě Android si lze prohlédnout v Příloze B.1. Další obrazovky, které aplikace nabízí, a nejsou zahrnuty v příloze:

- Zobrazení lékového záznamu pacienta pomocí ikony na liště seznamu nevydaných receptů (Obrazovka 4) nebo ve formuláři pro vyhledávání (Obrazovka 11)
- Zobrazení detailu výdejů v záložce výdeje na detailu receptu (Obrazovka 7)
- Vyhledávání receptu pomocí čárového kódu po stisku tlačítka Načíst čárový kód na obrazovce vyhledávání (Obrazovka 11)

Pro porovnání grafického zážitku lze v příloze B.2 prohlédnout vzorový průchod apliakcí pro lékaře na platformě iOS. Obrazovky a funkce, které nejsou zahrnuty v průchodu:

- Nastavení podepisovacího certifikátu z externího úložiště pomocí přepínače (Obrazovka 4)
- Zadávání doplňujících údajů o pacientovi (Obrazovka 2) při rozkliknutí tlačítka Zobrazit více
- Zobrazení detailu výdejů v záložce výdeje na detailu receptu (Obrazovka 10)
- Přidání položky na recept (Obrazovka 12)
- Možnost smazat vyplněné údaje pomocí ikony popelnice (Obrazovka 2)
- Zobrazení nápovědy z menu (Obrazovka 3)
- Vytvoření receptu pomocí ikony diskety (Obrazovka 2) nebo na konci zadávání údajů o předepisujícím a doporučujícím lékaři na záložce Lékař (Obrazovka 11)

Kapitola 7

Publikování aplikace na GooglePlay a AppStore

Publikování aplikací na App store je jeden z posledních kroků celého vývoje aplikace. Pro vystavení aplikací veřejnosti mají obě platformy své internetové obchody, ve kterých je aplikace dostupná ke stažení. Android distribučních obchodů mnoho, neboť si jej může za určitých podmínek zřídit každý – vytvořené balíčky APK pak jsou distribuovány přes tento obchod – oficiální a největší obchod je **Google Play**, na který byly mobilní aplikace nasazeny. Naproti tomu Apple má pouze jeden oficiální kanál, a to je **App Store**. Vytvořené aplikace byly publikovány pod účtem Státního ústavu pro kontrolu léčiv, který má výhradní právo k jejich distribuci.

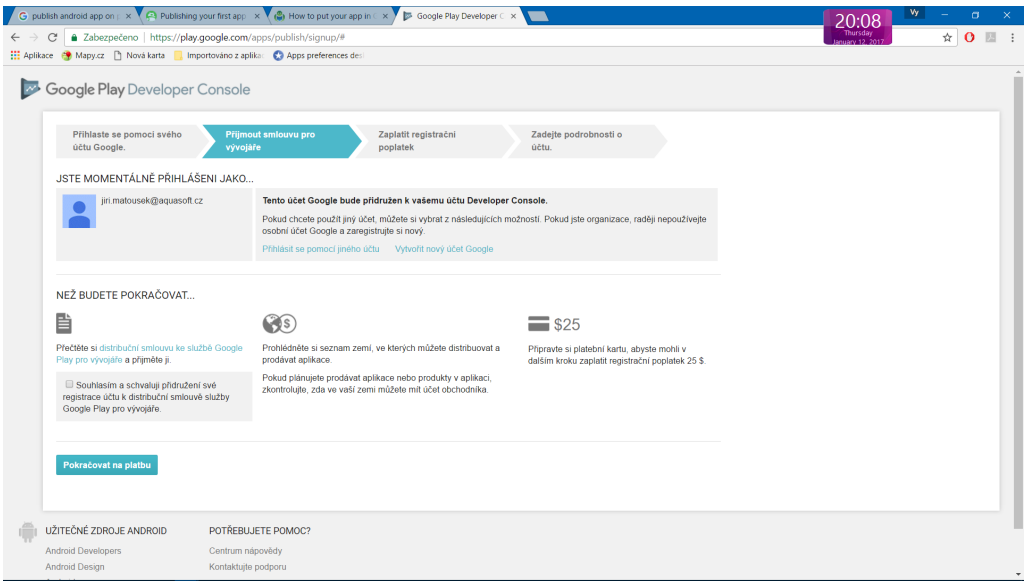
7.1 Publikování na Google play

Pro publikování mobilních aplikací na Google Play je potřeba zaregistrovat účet na stránkách Google Play Publisher. Součástí této jednorázové registrace je i zaplacení jednorázového poplatku ve výši $ **25**.

7.1.1 Založení Google Play Publisher účtu

1. Registrace se provede na URL
   https://play.google.com/apps/publish/signup/

2. Pro účely publikování mobilní aplikace je nutné vytvořit účet (obrázek 7.1) na Google Play pro organizaci, která vlastní tuto mobilní aplikaci, včetně nastavení platební karty pro platby.



Obrázek 7.1: Průvodce registrace vývojářského účtu

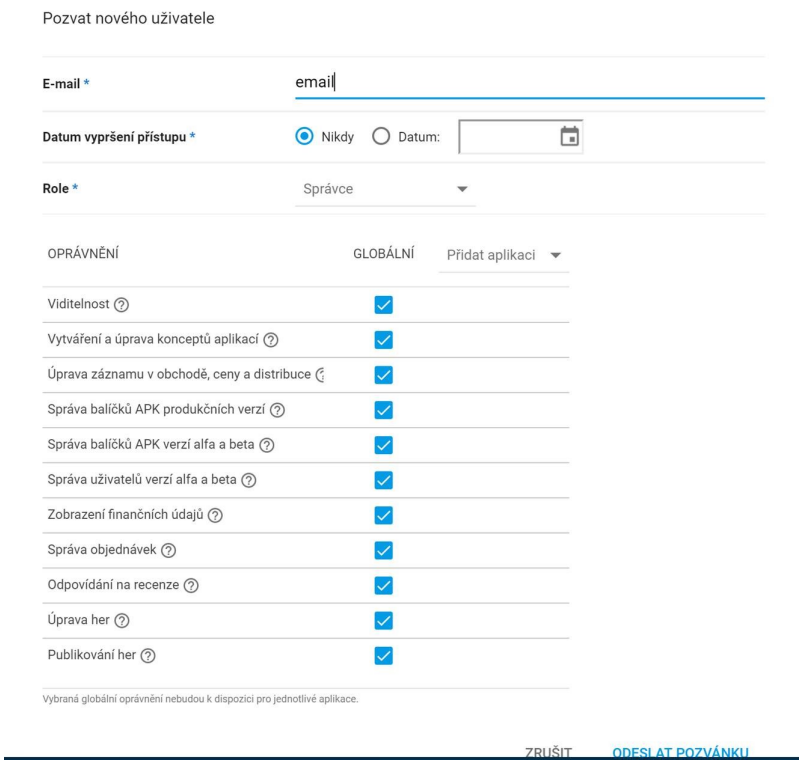
3. Po registraci účtu a přihlášení je dalším krokem zaplacení jednorázového poplatku $ **25** pomocí zaregistrované platební karty.

4. Následně je možné vyplnit informace o identitě – název organizace, funkční emailovou adresu apod.

5. Potvrzení registrace může trvat až 48 hodin.

7.1.2 Vytvoření přístupu vzdáleného rozhraní

Pro nasazování na Google play je zapotřebí v Google play Console v sekci **Nastavení->uživatelé a oprávnění->Pozvat nového uživatele** s rolí Správce, oprávnění - vše (obrázek 7.2



Obrázek 7.2: Vytvoření přístupu vzdáleného rozhraní

Pro jednoduché nahrávání nových verzí jsou zapotřebí následující kroky:

1. Nastavení přístupu k rozhraní API vytvořit Klient OAUTH pro publikování aplikace prostřednictvím webové služby (7.3).

2. Získání klientských údajů pro API pro připojení na Google Play (7.4)

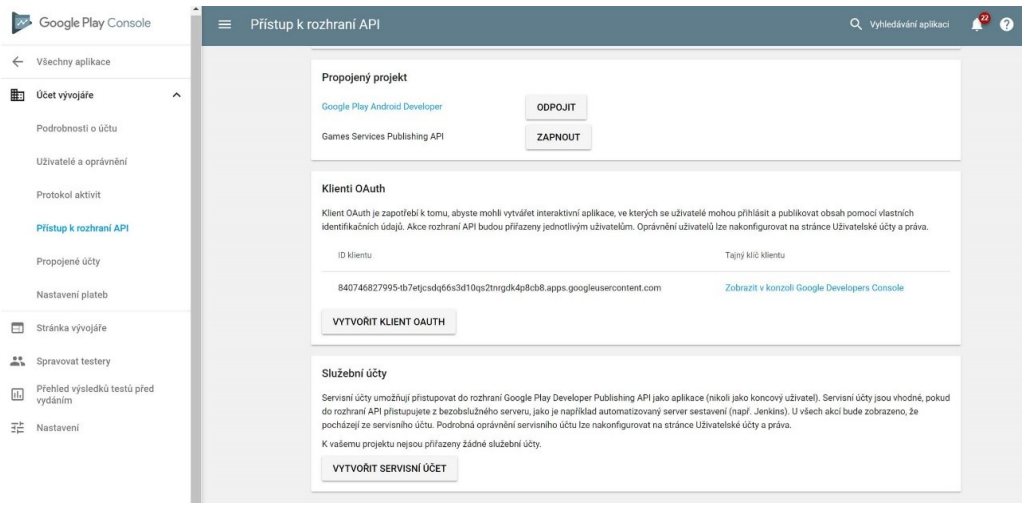
3. Ve Visual Studio 2017 nad projektem kliknout na **Archive** (7.5)

4. Poté, co se projekt zkompiluje nad vybraným řádkem zvolit volbu Distribute

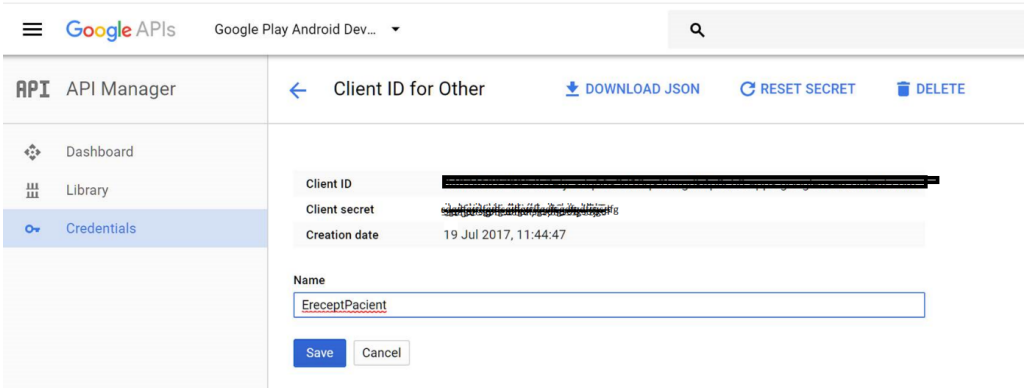
5. Výběr kanálu Google Play (7.8)

6. Registrace Google Play Account pomocí tlačítka Přidat (7.6)

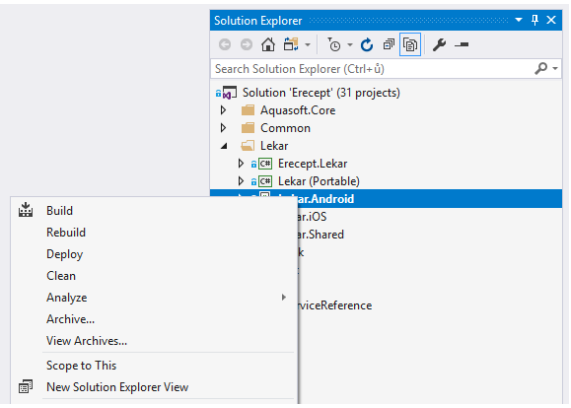
7. Vyplnění údajů z kroku 2 do formuláře při zakládání nového účtu (7.7)



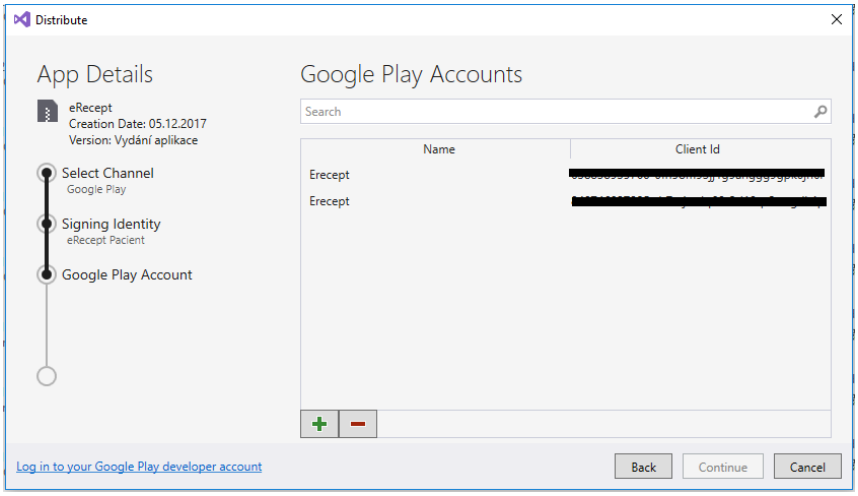
Obrázek 7.3: Nastavení přístupu k rozhraní Klient OAUTH



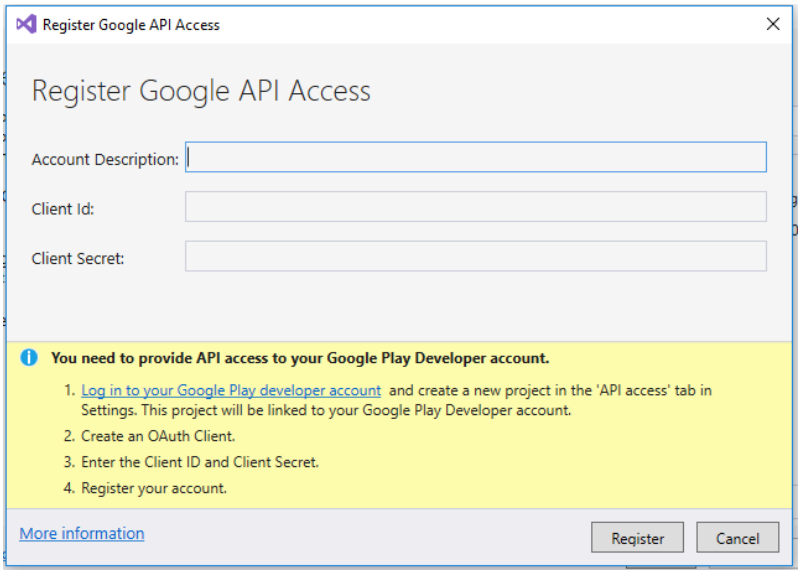
Obrázek 7.4: Získání přihlašovacích údajů ke službě Google Play Developer



Obrázek 7.5: Archivace Android projektu v IDE Visual Studio 2017



Obrázek 7.6: Přehled stávajících účtů Google Play – možnost přidání

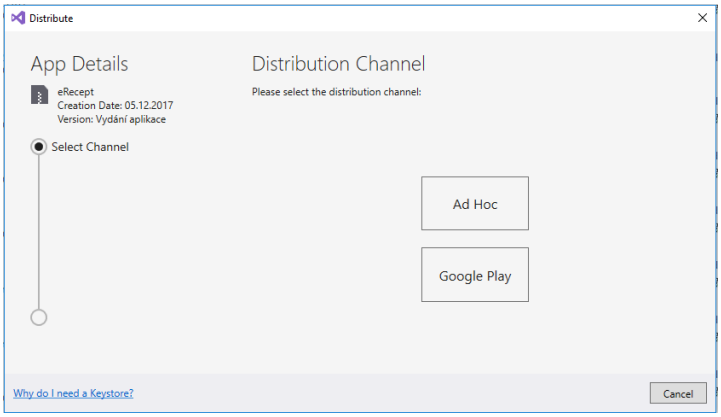


Obrázek 7.7: Vytvoření nového účtu Google API Access

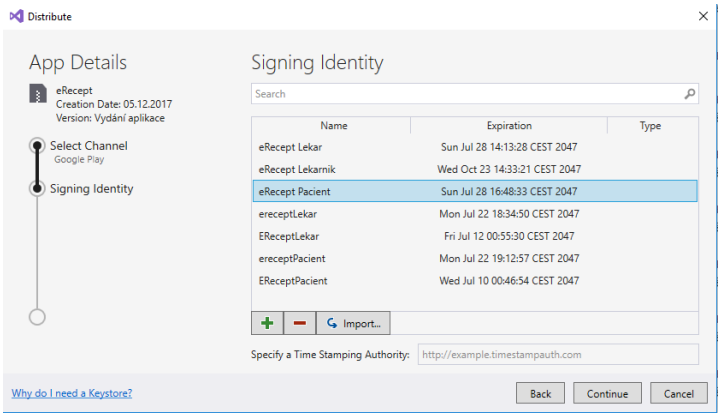
7.1.3 Vytvoření podepisovacího certifikátu

Všechny aplikace umístěné na Google Play musí být podepsány vlastním podeisovacím certifikátem tzv **Android KeyStore**, a poté všechny verze. Klíč je možné buď vytvořit externím nástrojem např. v Android Studio, nebo přímo ve Visual Studio 2017 IDE. Stejným procesem Archivace, jako je uveden v předchozí kapitole lze vytvořit tzv. **Signing identity**:

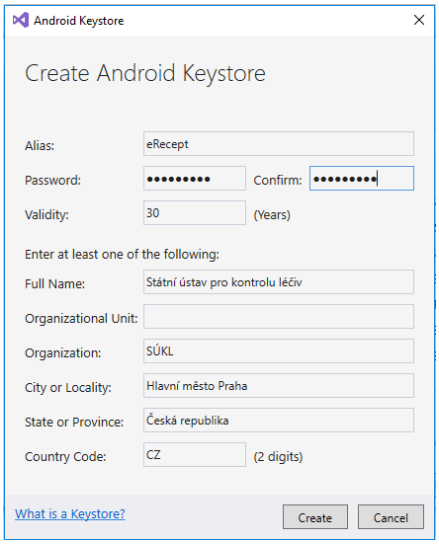
1. Na projektu zvolit archivace (7.5)
2. Po zkompilování projektu vybrat řádek a akci **Distribute** (7.8)
3. Vybrat přidat Podepisující identitu pomocí tlačítka plus (7.9)
4. Vyplnit formulář pro založení nového podepisovacího Certifikátu Android KeyStore (7.10)



Obrázek 7.8: Výběr distribučního kanálu



Obrázek 7.9: Seznam dostupných podepisovacích certifikátů, s možností přidat nový

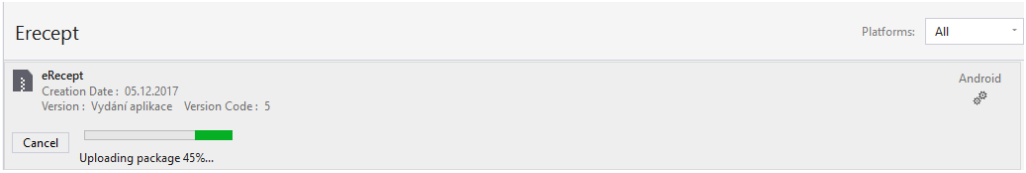


Obrázek 7.10: Vytvoření Android Keystore pro podepisování aplikací

7.1.4 Vystavení aplikace na Google Play

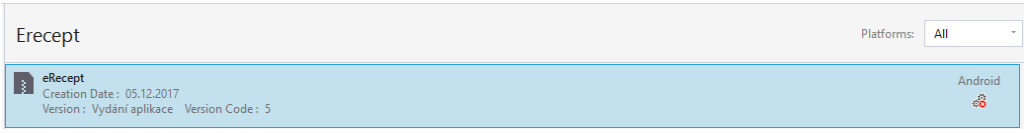
Díky předešlému nastavení je již samotné vystavení z větší části možné skrze Visual Studio 2017.

1. Sestavení Android projektu pomocí Rebuild akce
2. Archivace projektu pomocí kontextového menu
3. Zvolení akce Distribute, které vyvolá průvodce nasazení aplikace
4. Zvolení distribučního kanálu Google Play
5. Zvolení správné podepisovací identity
6. Zvolení vytvořeného Google API účtu
7. Zadání hesla k certifikátu
8. Zvolení větve Production pro nahrání aplikace do produkce
9. Zmáčknutí tlačítka Upload

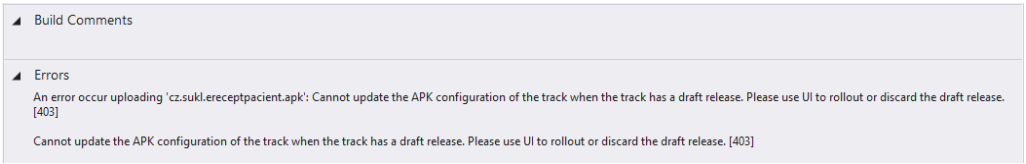


Obrázek 7.11: Průběh nahrávání aplikace na Google Play

10. Během nahrávání může dojít k chybě, která je značena červenou ikonkou křížku u vybraného distribuovaného řádku



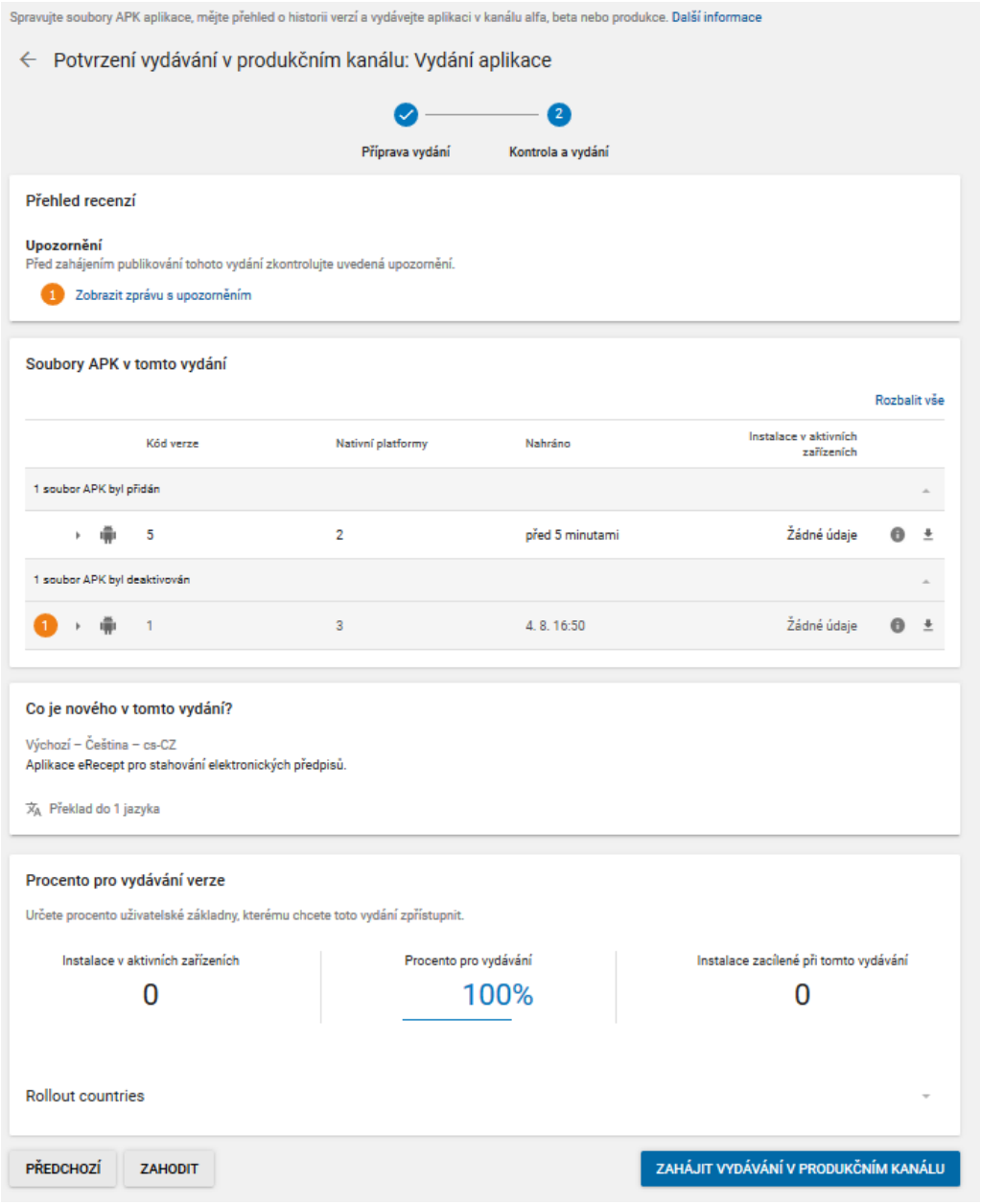
Obrázek 7.12: Neúspěch nasazení aplikace do produkce



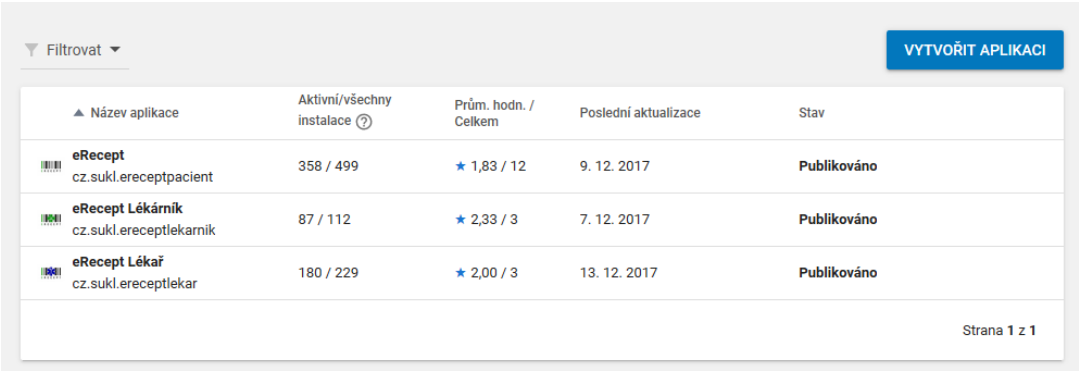
Obrázek 7.13: Informační hlášení s důvodem neúspěchu nasazení aplikace

11. Pakliže se nasazení povedlo v pořádku, posledním krokem nasazení je na webu https://play.google.com/apps/publish/ ve správě vydání aplikace **Zahájit vydávání v produkčním kanálu** (7.14)

Jakmile dojde na straně serveru k úspěšnému zpracování nové verze (ať už počáteční nebo každé další, na hlavní straně v Google Play Console bude u nasazené aplikace textový popisek **Publikováno** (viz. 7.15).



Obrázek 7.14: Obrazovka s potvrzením zahájení vydávání nové verze v produkčním prostředí



Obrázek 7.15: Všeobecný přehled o stavu aplikací v Google Play Console

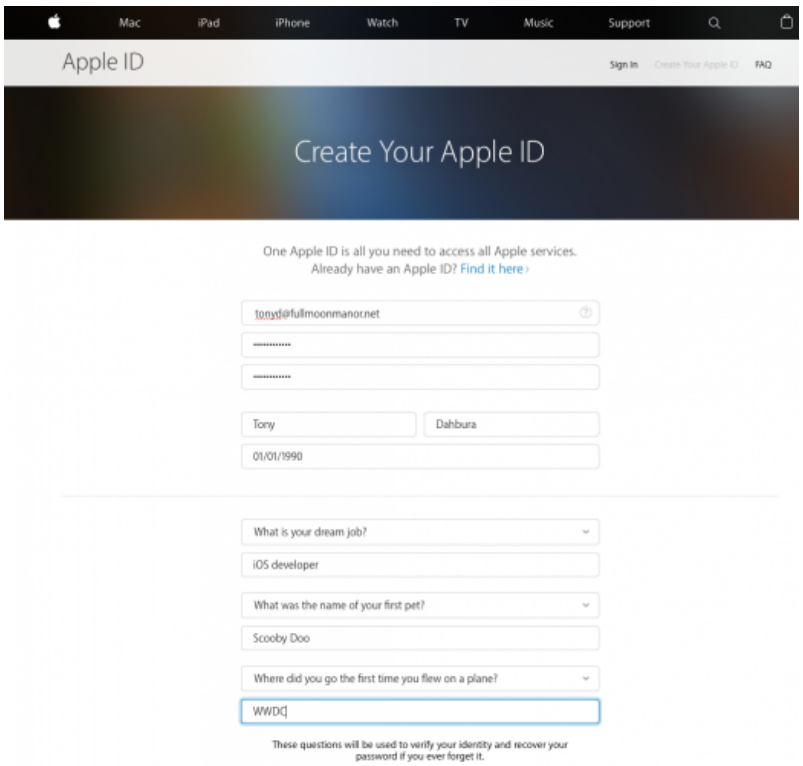
7.2 Publikování na AppStore

Tato kapitola obsahuje soupis požadavků a kroků potřebných k tomu, aby mohla být aplikace pro mobilní zařízení běžící na platformě iOS zveřejněna (publikována) na oficiálním úložišti App Store firmy Apple. Celý proces publikace aplikace se skládá z několika dílčích nutných kroků, které jsou popsány v následujících kapitolách.

7.2.1 Registrace na Apple Developer Site

Pro úspěšnou publikaci aplikace pro iOS je nutné mít zaregistrován účet na Apple Developer Site. Registrace se provádí pouze jednorázově, pokud ještě žádné Apple ID není zaregistrováno společností, která vlastní mobilní aplikaci. Pro další následné publikace (například aktualizací aplikace) se použije již takto vytvoření Apple ID účet.

- Registrace se provádí na webových stránkách https://developer.apple.com/ , volba „Account“.
- Vytvoření nového Apple ID a registrace. Je potřeba vyplnit funkční emailovou adresu, heslo a další bezpečnostní informace.
- Pro úspěšné dokončení nové registrace je nutné vyplnit ověřovací kód, který byl odeslán na uvedenou emailovou adresu.
- Po dokončení registrace se lze přihlásit k vytvořenému Apple ID účtu.



Obrázek 7.16: Obrazovka webové aplikace pro založení Apple ID

7.2.2 Apple Developer Program

Aby bylo možné publikovat aplikaci na App Store, je nutné připojit se k Apple Developer Program. Tato registrace je platná pouze na jeden rok a je nutné ji pravidelně obnovovat za poplatek $ **99,00 ročně.**

1. Nejprve je nutné se přihlásit k účtu na Apple Developer Site na webové adrese https://developer.apple.com/

2. Na domovské stránce zvolením odkazu Join the Apple Developer Program se zahájí proces přihlášení k Apple Developer Program.

3. Je nutné zvolit roli, v jaké se přihlášení provádí: jako jednotlivec nebo jako organizace.

4. Dále je potřeba vyplnit kontaktní informace o organizaci

5. Je možné nastavit Automatické každoroční obnovení členství v Apple Developer Program.

6. Jako poslední krok je nutné zaplatit roční poplatek $ 99,00, pomocí platební karty. Tento poplatek je nutné zaplatit po každém ročním obnovení členství.

7.2.3 Certifikáty, ID a profily

Každá aplikace pro platformu iOS musí obsahovat podepsaný Apple certifikát, proto je nutné jej nejdříve vygenerovat. Rovněž aplikace musí mít vygenerované ID, aby ji bylo možné publikovat na App Store. Toto vše je možné vygenerovat, pokud je vytvořen správný distribuční profil.

Pro úspěšné vytvoření profilu je třeba mít nainstalovanou aplikaci *Xcode*, vývojové prostředí firmy Apple, na počítači Mac. Aplikace Xcode je zpřístupněna na oficiálním Mac App Store firmy Apple.

7.2.3.1 Vygenerování produkčního certifikátu aplikace

1. Nejprve je nutné se přihlásit k účtu na Apple Developer Site na webové adrese https://developer.apple.com/

2. Přejít do sekce Certificates, IDs & Profiles.

3. Přidat certifikát pro iOS, tvOS, watchOS, volba pro Production/App Store and Ad Hoc.

4. Pro vlastní vygenerování certifikátu je třeba nejprve vytvořit Požadavek na certifikát v programu Keychain Access, nainstalovaný na počítači Apple. V tomto programu se vytvoří a uloží na disk soubor požadavku k certifikátu.

5. Soubor s požadavkem k certifikátu se použije na Apple Developer Site k vygenerování vlastního certifikátu.

6. Vygenerovaný certifikát uložit a nainstalovat pomocí aplikace Keychain Access.

7. Vytvoření ID aplikace

8. Každá aplikace potřebuje mít svoje App ID. Přejít do sekce Identifiers\ App IDs.

9. Zvolit Přidat App ID

10. Vyplnit popis ID, předponu a příponu pro Explicit App ID a potvrdit formulář

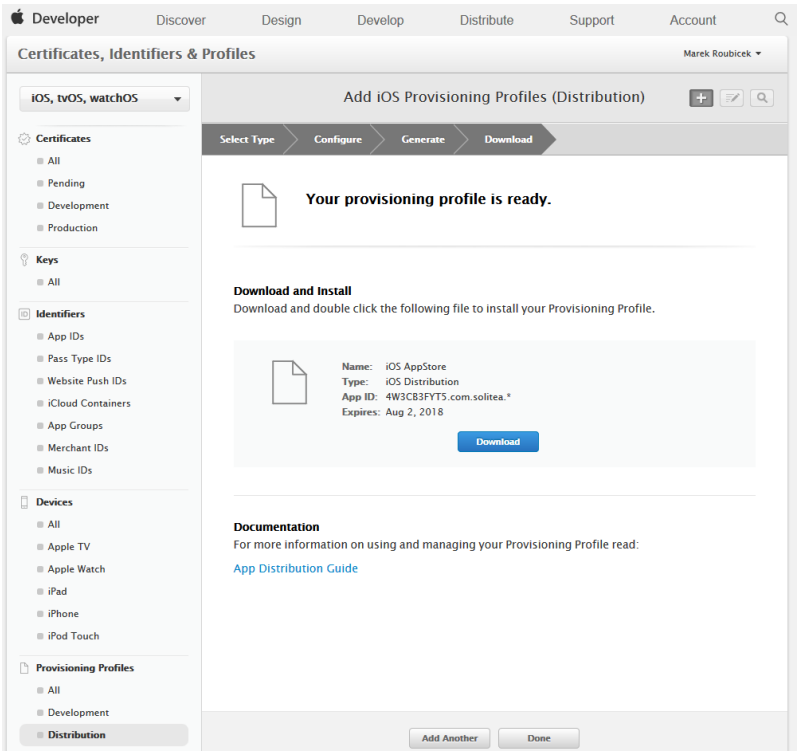
7.2.3.2 Vytvoření profilu

1. Přejít do sekce Provisioning Profiles\ All

2. Zvolit Přidat profil

3. Vybrat Distribution / App Store. Pro nový profil musí být použit App ID a certifikát vytvořené v předchozích krocích.

4. Vytvořený profil je potřeba stáhnout (soubor viz. 7.17) a otevřít v aplikaci Xcode na počítači Mac. Tento vytvořený profil je nutné pro sestavení mobilní aplikace (IPA balíček) pro její publikaci.

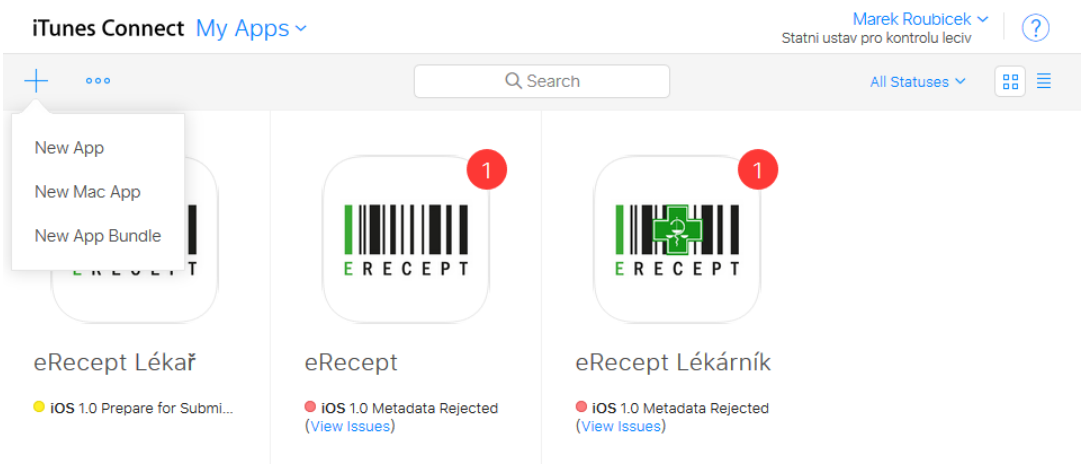


Obrázek 7.17: Přehled vytvořeného provisioning profulu pro distribuci iOS aplikací

7.2.4 iTunes Connect

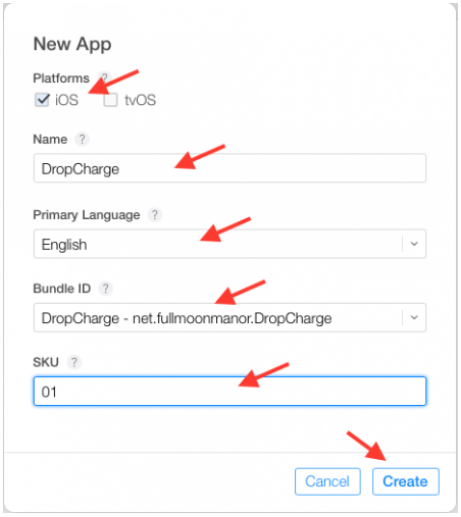
Předposledním krok pro publikaci mobilní aplikace na App Store je její registrace na iTunes Connect, což je webová aplikace pro správu mobilních aplikací. Je potřeba provést následující kroky:

1. Na webové stránce https://developer.apple.com/membercenter se přihlásit pod iOS Delevoper účtem
2. Zvolit iTunes Connect odkaz.
3. Na hlavní stránce iTunes Connect, otevřít sekci My Apps.
4. Přidat novou aplikaci volbou menu New App. (7.18)



Obrázek 7.18: Vytvoření nové aplikační záložky ve správě Itunes Connect

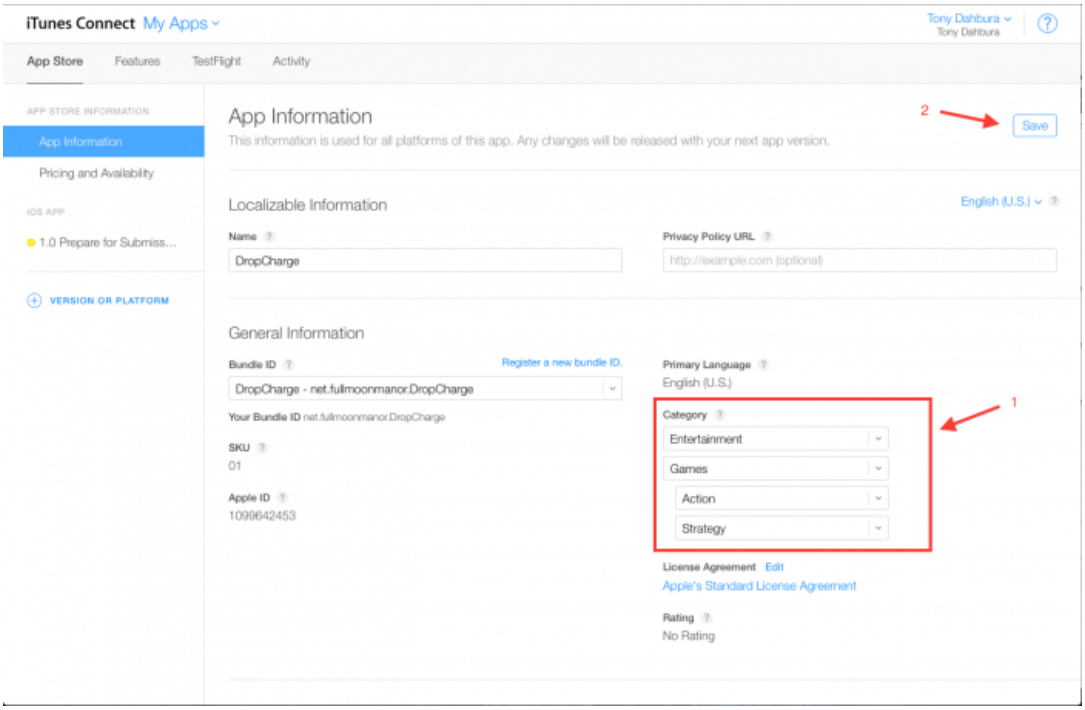
5. Na následujícím formuláři (7.19) je třeba vyplnit položku Platforma iOS, název aplikace, primární jazyk, Bundle ID (registrované ID v předchozích krocích), SKU (volitelné ID).



Obrázek 7.19: Konfigurace pro novou aplikaci v Itunes Connect

6. Potvrdit vytvoření aplikace

7. Dále je na následující obrazovce potřeba vyplnit Kategorie a uložit.



Obrázek 7.20: Správa aplikace v Itunes Connect (před nasazením na AppStore)

8. Dále zvolit sekci *Pricing and Availability*. Zde se nastaví, že mobilní aplikace bude přístupná zdarma koncovým uživatelům.

9. Dále zvolit sekci *Prepare for Submission*. V této sekci je nutné nadefinovat a nahrát různé komponenty mobilní aplikace vyžadované pro její úspěšnou publikaci. Jedná se o ikonu aplikace a fotky aplikace pro různá cílová zařízení (iPhone, iPad apod.), popis aplikace a zadání klíčových slov, pro která bude aplikace nabízena, dále pak např., URL webové stánky pro podporu aplikace.

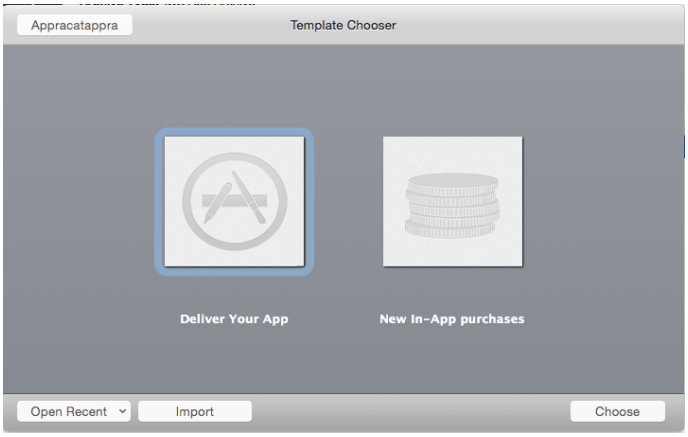
10. Je nutné vyplnit sekci *Rating* v rámci sekce *Prepare for Submission.*

11. Dále se třeba vyplnit sekci *App Review Information.* V této sekci se zadávají informace pro ověřovatele a schvalovatele mobilní aplikace na straně firmy Apple. Je zde potřeba zadat stručný popis workflow aplikace pro testování včetně funkčních přístupových údajů (jméno, heslo pro přístup do mobilní aplikace). Zadat je třeba i kontaktní údaj, aby mohlo dojít ke kontaktu schvalovatelem v případě nejasností.

7.2.5 Vystavení aplikace na App Store

Poslední krok je vlastní vystavení mobilní aplikace na App Store, kde dojde k její ověření a schválení firmou Apple. Pro vystavení aplikace je potřeba použít aplikaci *Application Loader* na Mac počítači. Tato aplikace je ke stažení na https://itunesconnect.apple.com/apploader/ApplicationLoader_3.0.dmg.

1. Otevřít aplikaci *Application Loader* a zvolit volbu Deliver Your App. (7.21)



Obrázek 7.21: Okno aplikace Application Loader, pro nahrání IPA souboru na App Store

2. Vybrat IPA soubor reprezentující mobilní aplikaci. IPA soubor bude dodán dodavatelem.

3. V dalších krocích průvodce dojde postupně k validaci a nahrání aplikace na App Store.

Závěr

Cílem práce bylo prozkoumat možnosti multiplatformního vývoje a na základě poznatků a požadavků Státního ústavu pro kontrolu léčiv vytvořit vlastní mobilní aplikace pro pacienty a lékaře umožňující vytváření a nahlížení na elektronické recepty. Z tohoto pohledu byly cíle naplněny, neboť systém byl úspěšně spuštěn v listopadu 2017 a během prvních týdnů ostrého provozu si aplikaci pro lékaře, pacienty i lékárníky stáhly již dva tisíce uživatelů přes obchod Google Play.

V době psaní této práce se nepovedlo nasadit vyvinutou aplikaci na App Store (pouze do mezistupně na Itunes Connect), jelikož na rozdíl od Google, společnost Apple podrobuje aplikace ručnímu testování jejich testery, a v případě, že aplikace vyžaduje přihlašovací údaje, musí jim je vývojář poskytnou proti produkčnímu prostředí. Ošetření, aby tester Apple omylem nebo úmyslně nevytvořil nechtěná data se musí udělat na straně serveru. V době psaní této práce nebyl takový účet ještě zrealizován, takže aplikace pro platformu iOS budou nasazeny, jakmile dojde k řádnému otestování společností Apple.

Práce zároveň čtenáře seznamuje s celým fungováním systému eRecept, jehož jsou mobilní aplikace součástí. Z prvních recenzí musím konstatovat, že aplikace, zejména pacientská, získává spíše negativní reakce - a to zejména z toho důvodu, že systém eRecept je prvním, který využívá elektronické identity, a způsob registrace pro přístup k základním registrům prostřednictvím státních systémů není úplně jednoduchý. Z tohoto pohledu věřím v lepší pochopení koncovými uživateli a jejich trpělivost při překonání registračních překážek, neboť systém eReceptu v mobilu nabízí spíše pozitiva.

Z pohledu technologie Xamarin a multiplatformního vývoje, tato práce nastiňuje zásadní rozdíly mezi jednotlivými přístupy a v případě Xamarinu poskytuje dostatečné informace k tomu, aby na základě přečtení mohl čtenář vyvinout vlastní aplikace a zároveň chápal jeho principy. Jelikož je komunita kolem Xamarinu anglicky komunikující, může tato práce posloužit jako návod pro čtenáře, kteří angličtinu neovládají na takové úrovni.

Díky této práci jsem získal mnoho znalostí a zkušeností v oblasti mobilního vývoje (ne jen ohledně Xamarinu), které bych rád zužitkoval i v budoucím profesním životě. Vzhledem ke stále vysoké poptávce po vývojářích mobilních aplikací věřím, že se mi tyto dovednosti budou hodit.

Literatura

[1] Kříž, L.: Do mobilních zařízení vloni zamířilo 156 miliard aplikací. 12. 5. 2016, [cit 2017-11-15]. Dostupné z: http://www.businessit.cz/cz/do-mobilnich-zarizeni-vloni-zamirilo-156-miliard-aplikaci.php

[2] Mikudík, R.: Android s iOS drtí mobilní svět. Ostatní jsou v klinické smrti. 26. 2. 2016, [cit 2017-11-15]. Dostupné z: http://mobil.idnes.cz/android-s-ios-drti-mobilni-svet-dmu-/mob_tech.aspx?c=A160225_194408_mob_tech_ram

[3] AppBrain: Most popular Google Play categories. 2017, [cit 2017-11-15]. Dostupné z: https://www.appbrain.com/stats/android-market-app-categories

[4] ČTK: Česko je až padesáté na světě podle elektronizace veřejné správy. 4. 8. 2016, [cit 2017-11-15]. Dostupné z: https://zpravy.aktualne.cz/ekonomika/cesko-je-az-padesate-na-svete-podle-elektronizace-verejne-sp/r~02c5b38c5a5111e6a3e5002590604f2e/

[5] statista.com: Most popular Apple App Store categories in October 2017. 2017, [cit 2017-12-30]. Dostupné z: https://www.statista.com/statistics/270291/popular-categories-in-the-app-store/

[6] Ford, K.: Mobile Development Platform Performance. 7. 4. 2015, [cit 2017-12-30]. Dostupné z: https://magenic.com/thinking/mobile-development-platform-performance

[7] Václavík, J.: Jak vyvíjet mobilní aplikace. 4. 6. 2015, [cit 2017-11-16]. Dostupné z: http://janvaclavik.cz/jak-vyvijet-mobilni-aplikace/

[8] Unity: Unity - Multiplatform - Publish your game to over 25 platforms. 2017, [cit 2017-11-16]. Dostupné z: https://unity3d.com/unity/features/multiplatform

[9] The Qt Company: Mobile App Development with Qt. 2017, [cit 2017-11-16]. Dostupné z: https://www.qt.io/mobile-app-development/

[10] AltexSoft: The Good and The Bad of Xamarin Mobile Development. 21. 11. 2017, [cit 2017-12-30]. Dostupné z: https://www.altexsoft.com/blog/mobile/the-good-and-the-bad-of-xamarin-mobile-development/

[11] Ford, K.: Mobile Development Platform Performance Part 2. 22. 4. 2015, [cit 2017-12-30]. Dostupné z: https://magenic.com/thinking/mobile-development-platform-performance-part-2-native-cordova-classic-xamarin-xamarin-forms

[12] Mono Project: History. 2018, [cit 2017-11-15]. Dostupné z: http://www.mono-project.com/docs/about-mono/history/

[13] de Icaza, M.: The Mono Project [video]. 8.12.2009, [cit. 2017-11-15]. Dostupné z: https://channel9.msdn.com/Events/MIX/MIX10/EX02

[14] Xamarin Inc.: AOT Building process. Březen 2017, [cit 2017-11-15]. Dostupné z: https://developer.xamarin.com/guides/ios/under_the_hood/architecture/

[15] Xamarin Inc.: Basic overview of Xamarin.iOS architecture. Březen 2017, [cit 2017-11-15]. Dostupné z: https://developer.xamarin.com/guides/ios/under_the_hood/architecture/

[16] Xamarin Inc.: Basic overview of Xamarin.Android architecture. Březen 2017, [cit 2017-11-15]. Dostupné z: https://developer.xamarin.com/guides/android/under_the_hood/architecture/

[17] Xamarin Inc.: Shared Projects. 2017, [cit 2017-11-15]. Dostupné z: https://developer.xamarin.com/guides/cross-platform/application_fundamentals/shared_projects/

[18] Xamarin Inc.: Portable Class Libraries. 2017, [cit 2017-11-15]. Dostupné z: https://developer.xamarin.com/guides/cross-platform/application_fundamentals/pcl/introduction_to_portable_class_libraries/

[19] Xamarin Inc.: .NET Standard. 2017, [cit 2017-11-15]. Dostupné z: https://developer.xamarin.com/guides/cross-platform/application_fundamentals/net-standard/

[20] Google, Inc.: The Activity Lifecycle. 2017, [cit 2017-11-15]. Dostupné z: https://developer.android.com/guide/components/activities/activity-lifecycle.html

[21] Apple Inc.: The App Life Cycle. 27. 3. 2017, [cit 2017-11-15]. Dostupné z: https://developer.apple.com/library/content/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/TheAppLifeCycle/TheAppLifeCycle.html

[22] Dajbych, V.: MVVM: Model-View-ViewModel. 21. 04. 2009, [cit 2017-12-31]. Dostupné z: https://www.dotnetportal.cz/clanek/4994/MVVM-Model-View-ViewModel

[23] Saleh, H.: MVVM architecture, ViewModel and LiveData (Part 1). 31. 05. 2017, [cit 2017-12-31]. Dostupné z: https://proandroiddev.com/mvvm-architecture-viewmodel-and-livedata-part-1-604f50cda1

[24] Xamarin Inc.: Navigation. 2017, [cit 2017-12-31]. Dostupné z: https://developer.xamarin.com/guides/xamarin-forms/application-fundamentals/navigation/

[25] Xamarin Inc.: Layouts. 2017, [cit 2017-12-31]. Dostupné z: https://developer.xamarin.com/guides/xamarin-forms/user-interface/layouts/

[26] Xamarin Inc.: Introduction to DependencyService. 2017, [cit 2017-12-31]. Dostupné z: https://developer.xamarin.com/guides/xamarin-forms/application-fundamentals/dependency-service/introduction/

[27] CZ.NIC, z. s. p. o.: Jak na Internet - eGovernment. 2017, [cit 2017-12-31]. Dostupné z: https://www.jaknainternet.cz/page/1718/egovernment/

[28] Správa základních registrů : eIdentita.cz. 2010 – 2018, [cit 2017-12-31]. Dostupné z: http://www.szrcr.cz/pro-media/eidentita-cz

[29] Správa základních registrů : Příručka k využití služeb národní identitní autority pro poskytovatele služeb veřejné správy. 2010 – 2018, [cit 2017-12-31] verze 0.8. Dostupné z: http://www.szrcr.cz/uploads/Dokumenty/SeP_pr_i_ruc_ka_SZR_0v8_2017_03_31.pdf

[30] Přívratský, T.: Centrální úložiště receptů pro SÚKL. 14. 2. 2011, [cit 2017-12-31]. Dostupné z: http://www.itpoint.cz/aquasoft/clanky/?i=centralni-uloziste-receptu-pro-sukl-6588

[31] Státní ústav pro kontrolu léčiv: Centrální úložiště receptů pro SÚKL. 2017, [cit 2017-12-31]. Dostupné z: https://www.epreskripce.cz/sites/default/files/soubory/vzor_pruvodka_2xlp.pdf

[32] Březovský, P.: eRecept. 30. 09. 2017, [cit 2017-12-31]. Dostupné z: http://slideplayer.cz/slide/1943581/

[33] Státní ústav pro kontrolu léčiv: Registrace a aktivace uživatelského profilu k přístupu do systému eRecept pro pacienta. 2017, [cit 2017-12-31]. Dostupné z: https://www.epreskripce.cz/sites/default/files/soubory/sukl_manual_pristup_nia_v02.pdf.pdf

[34] Debef, F.: Návod na aktivaci účtu SÚKL a vygenerování certifikátu pro přístup do centrálního úložiště receptů. 06. 11. 2017, [cit 2017-12-31]. Dostupné z: https://www.epreskripce.cz/sites/default/files/navod_erp_lekari_20171106_0.pdf

[35] Odbor IT, SÚKL: Postup při zřízení účtu lékárny a přidělení přihlašovacích údajů lékárny k centrálnímu úložišti elektronických receptů. 13. 6. 2016, [cit 2017-12-31]. Dostupné z: http://www.sukl.cz/lekarny/postup-pri-zrizeni-uctu-lekarny-a-prideleni-prihlasovacich

[36] Solitea Business Solutions s.r.o.: SÚKL - Informační systém e-Recept. Prováděcí projekt, Solitea Business Solutions s.r.o., 10. 1. 2017.

[37] arstechnica: Xamarin studio on Windows. 2015, [cit 2017-12-31]. Dostupné z: https://cdn.arstechnica.net/wp-content/uploads/2013/02/xamarin-studio-component-use.png

[38] Bockover, A.: Introducing Workbooks & Inspector. 18. 11. 2016, [cit 2017-11-14]. Dostupné z: https://blog.xamarin.com/introducing-workbooks-inspector/

[39] Hechtel, E.: Mobile Device Emulator and Simulator vs Real Device. 25. 5. 2016, [cit 2017-11-14]. Dostupné z: https://saucelabs.com/blog/mobile-device-emulator-and-simulator-vs-real-device

[40] Sobejana, M.: Market Guide for Mobile Application Testing Services. Technická zpráva, Gartner, 29. 10. 2015.

[41] Intel Corporation: Android* on Intel Platforms - What is HAXM? 2017, [cit 2017-11-15]. Dostupné z: https://01.org/android-ia/q-and-a/what-haxm

[42] Apple Inc.: About Simulator. 2017, [cit 2017-11-15]. Dostupné z: https://developer.apple.com/library/content/documentation/IDEs/Conceptual/iOS_Simulator_Guide/Introduction/Introduction.html

Příloha **A**

Seznam použitých zkratek

API Application Programming Interface

CSS Cascading Style Sheets

CÚER Centrální úložiště elektronických receptů

GPS Global Positioning System

GSM Global System for Mobile Communications

GUI Graphical user interface

HTML HyperText Markup Language

IDE Integrated Development Environment

NIA Národní identitní autorita

OS Operating System

PDF Portable Document Format

QR Quick Response kód

SDK Software Development Kit

SOAP Simple Object Access Protocol

SÚKL Státní ústav pro kontrolu léčiv

TFS Team Foundation Service

UI User Interface

URL Uniform Resource Locator

XAML Extensible Application Markup Language

XML Extensible markup language

Příloha **B**

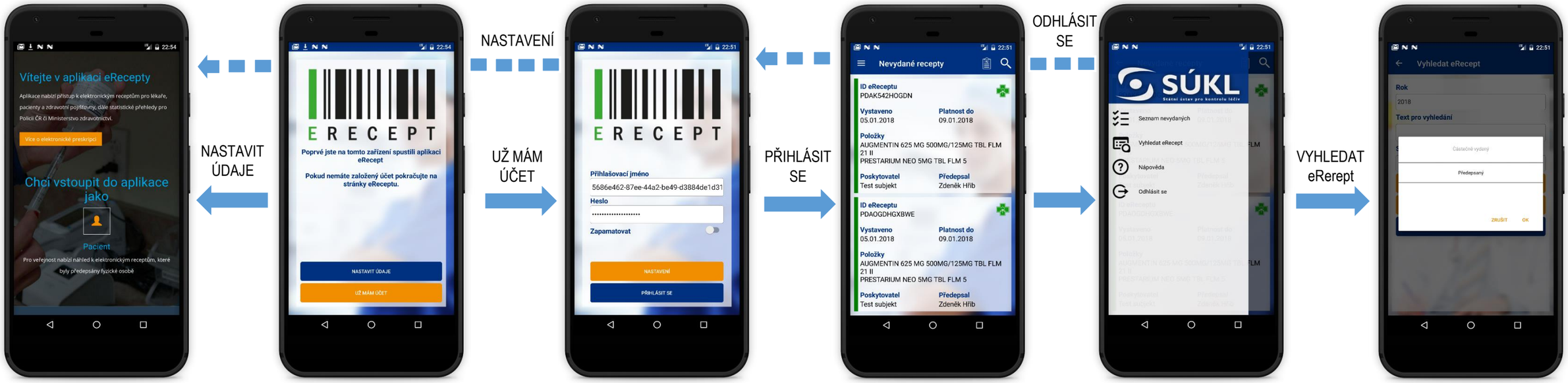
Seznam příloh

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl`
    - `TestApp` ..... zdrojové kódy, názornost projektů
    - `Concepts` ..... zdrojové kódy implementace konceptů Xamarin
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `plachtaPacient.pdf` ..... Příloha práce - grafická plachta průchodu aplikací pacient
  - `plachtaLekar.pdf` ..... Příloha práce - grafická plachta průchodu aplikací lékař

Příloha B.1 – Grafické znázornění zkráceného průchodu aplikací Pacient na platformě Android



3. Webová aplikace – pro nastavení IMEI k účtu

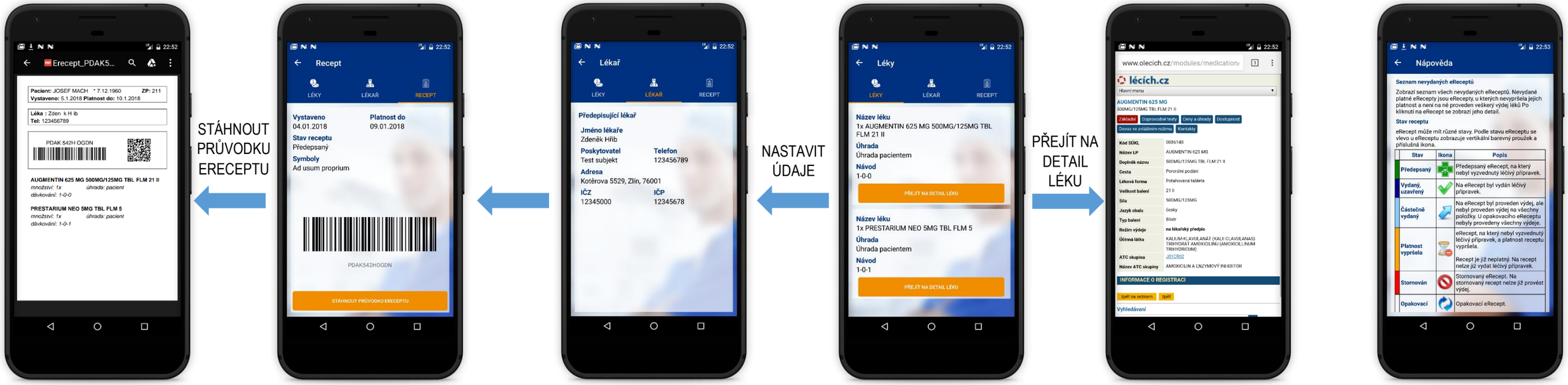
1. Úvodní obrazovka v případě prvního spuštění (bez provedeného úspěšného přihlášení)

2. Přihlašovací obrazovka

4. Seznam nevydaných receptů (stručné info)

8. Skrývací menu pomocí gesta swipe z levého okraje obrazovky

11. Vyhledávání receptů podle zadaných kritérií



10. PDF průvodka eReceptu

9. Detail receptu – informace o receptu s čárovým kódem

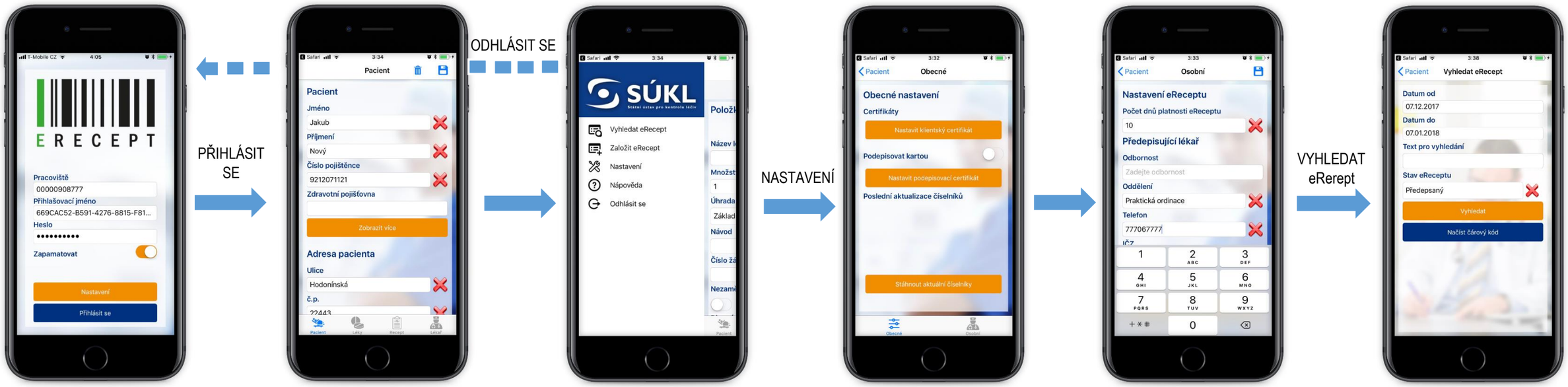
7. Detail receptu – Informace o předepisujícím lékaři

5. Detail receptu – seznam léků na receptu

6. Webový portál olecich.cz, s kompletními informacemi o vybraném léku

12. Nápověda k aplikaci

Příloha B.2 – Grafické znázornění zkráceného průchodu aplikací Lékař na platformě iOS



1. Přihlašovací obrazovka
Nastavení vyvolá nastavení klientského certifikátu

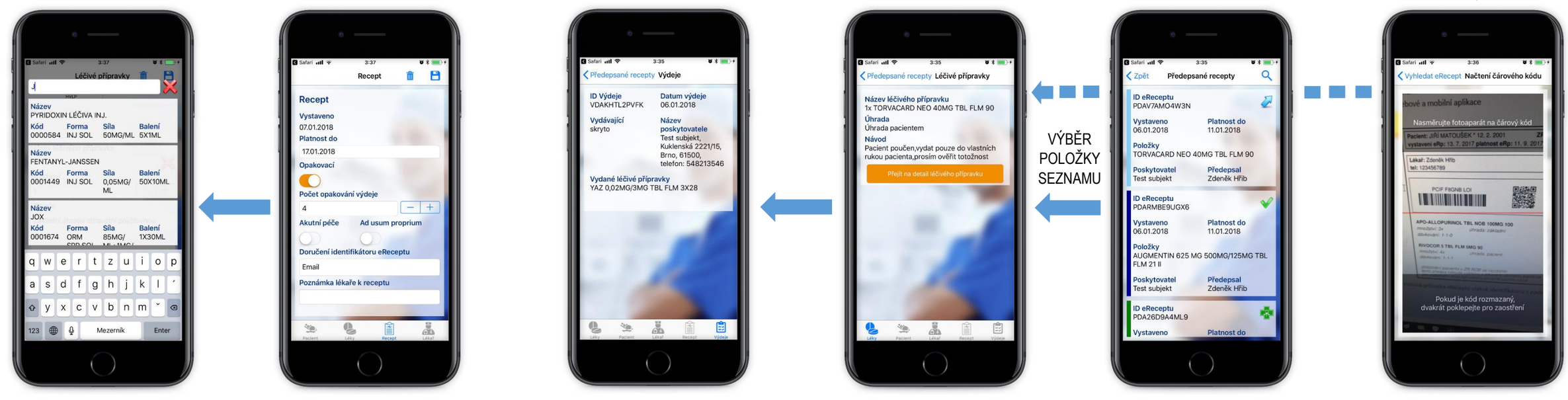
2. Založení eReceptu – Detail Pacienta s osobími údaji

3. Skrývací menu pomocí gesta swipe z levého okraje

4. Nastavení – nastavení podepisovacího certifikátu a stažení číselníků

5. Nastavení – osobní údaje předepisujícího lékaře pro předvyplňování

6. Vyhledávání receptů podle zadaných kritérií



12. Založení eReceptu – Výběr léčivého přípravku z číselníku

11. Založení eReceptu – nastavení údajů eReceptu

10. Detail receptu – Informace o výdejích eReceptu

9. Detail receptu – seznam léčivých přípravků na receptu

8. Seznam předepsaných receptů – stručné info

7. Vyhledávání eReceptu podle čárového kódu