ZÁPADOČESKÁ UNIVERZITA V PLZNI

FAKULTA EKONOMICKÁ

Bakalářská práce

Vývoj aplikací pro mobilní telefony

Application development for mobile phones

Lukáš Polívka

Plzeň 2016

Čestné prohlášení

Prohlašuji, že jsem bakalářskou práci na téma

„Vývoj aplikací pro mobilní telefony“

vypracoval samostatně pod odborným dohledem vedoucího bakalářské práce za použití pramenů uvedených v přiložené bibliografii.

Plzeň dne … …….

podpis autora

Poděkování

Tímto bych chtěl poděkovat vedoucímu práce doc. RNDr. Mikuláši Gangurovi, Ph.D. za jeho cenné rady a připomínky k problematice této práce.

Obsah

Úvod

V dnešní době všudypřítomných mobilních telefonů se aplikace pro tato zařízení stávají stále žádanější. Přestože každý chytrý telefon dokáže pracovat s webovými stránkami pomocí prohlížeče, jednoduchost a rychlost použití na míru zpracované aplikace je nesrovnatelná.

Cílem této práce je navrhnout a implementovat aplikaci pro organizaci zkouškového období, která umožní uživatelům systému STAG rychlou a jednoduchou organizaci studia bez nutnosti otevřít webový prohlížeč. Aplikace bude umožňovat zápis a odhlášení z termínu, naplánování zkouškových termínů a nabídne možnost upozornění na změnu obsazení vybraného termínu. Po vytvoření aplikace bude provedena pilotáž použití aplikace, jejíž výsledky budou analyzovány. Na základě výsledků pilotáže budou provedeny případné změny v aplikaci.

Mezi dílčí cíle práce patří srovnání populárních mobilních platforem a možnosti vývoje aplikací pro dané platformy. Na základě tohoto srovnání bude vybrán vhodný operační systém pro vývoj aplikace a bude provedena analýza vývojových prostředí a dalších nástrojů potřebných pro vývoj aplikace na vybrané platformě.

V první kapitole je provedena analýza existujících aplikací, které obsahují funkcionalitu podobnou navrhované aplikaci. První kapitola také obsahuje vlastní návrh aplikace. Ve druhé kapitole jsou popsány a srovnány tři nejrozšířenější operační systémy pro mobilní telefony, jejich výhody a nevýhody a je vybrána vhodná platforma pro vývoj aplikace. Ve třetí kapitole jsou popsána vývojová prostředí pro vybranou platformu a je v ní provedeno srovnání emulátorů.

Čtvrtá kapitola se zabývá vývojem aplikací pro vybraný operační systém. Popisuje architekturu aplikace, jejích komponent a prvků uživatelského rozhraní. V poslední kapitole je provedena implementace navržené aplikace a pilotáž použití. Kapitola dále obsahuje výsledky pilotáže a na jejich základě popisuje možnosti vylepšení aplikace.

1 Návrh aplikace

1.1 Analýza existujících aplikací

Pro zápis na zkouškové termíny studenti používají webovou aplikaci na Portálu. Ta je přizpůsobená k používání na počítači, nikoliv pro používání na malých dotykových displejích. Protože portál nenabízí žádnou verzi pro mobilní telefony, může být navigace na stránce v mobilním telefonu složitá a nepřehledná.

Na stránkách Informačního systému studijní agendy (IS/STAG) jsou k dohledání externí aplikace, které se systémem STAG pracují [1]. Jedinou aplikací, která obsahuje organizaci zkouškových termínů, je aplikace UniApps. Jedná se o aplikaci, která v sobě integruje mnoho funkcí IS/STAG. Umožňuje například zobrazení rozvrhu, jídelníčku v menze, informací o dopravě do školy, denního přehledu nebo právě přihlašování a odhlašování ze zkoušek. UniApps je dostupná pro Android, iOS a Windows Phone [2].

Problémem UniApps je fakt, že v aplikaci pro Android modul zkoušek nefunguje spolehlivě. Protože aplikace neumožňuje přihlášení na žádnou testovací databázi, nebylo skutečné otestování aplikace prakticky možné. Po přihlášení na Orion účet na ZČU však bylo zjištěno, že aplikace chybně zobrazuje stav dostupných termínů. Aplikace neumožňovala přihlášení na vybraný termín, protože ještě nebylo zahájeno zapisování. Ve skutečnosti však bylo možné se na termín přihlásit.

Modul zkouškových termínů v UniApps obsahuje pouze funkce, které existují ve webové aplikaci na Portálu a nenabízí žádnou rozšiřující funkcionalitu.

Mezi mobilní aplikace, které komunikují se systémem STAG, patří také aplikace STAG Rozvrh na Windows Phone a Primát.cz – Timetable pro Android. Obě uvedené aplikace zobrazují pouze rozvrh, neumožňují přihlašování na zkoušky a neobsahují žádné další funkce.

1.2 Návrh aplikace

Protože neexistuje vhodné a spolehlivé řešení pro organizaci zkoušek na mobilních telefonech, bude proveden návrh a následná implementace mobilní aplikace. Navržená aplikace by měla splňovat následující kritéria:

- Dostupnost pro co největší počet mobilních telefonů
- Dostupnost pro studenty všech škol, které používají IS/STAG

- Přehledné uživatelské rozhraní
- Rychlá a jednoduchá navigace v aplikaci

1.2.1 Funkce

Kromě standardních funkcí jako je zobrazení termínu a zápis na vybraný termín bude aplikace obsahovat nové funkce, které nejsou dostupné v žádné existující aplikaci. Novými funkcemi jsou plánovač zkouškových termínů a upozornění na změnu obsazení termínu. Tyto funkce, blíže popsané níže, by měly motivovat uživatele k používání aplikace, protože nabízí něco navíc a nejsou pouhou mobilní verzí Portálu.

- **Přihlášení studenta do systému –** Každý student se bude muset přihlásit ke svému účtu v systému STAG, aby mohl aplikaci používat. Aplikace bude umožňovat přihlášení studentům všech škol, které používají IS/STAG.
- **Zobrazení zapsaných zkouškových termínů –** Aplikace po spuštění zobrazí přehled zkouškových termínů, na které je student zapsán a poskytne informace o vybraných termínech, jako například místnost a čas konání zkoušky.
- **Zobrazení a filtrování všech dostupných termínů –** Aplikace nabídne možnost filtrování termínů podle data, předmětu, zkoušejícího a místnosti konání zkoušky. Filtrovací funkce by měla výrazně ulehčit vyhledání správného termínu, protože uživatel nebude nucen listovat přes dlouhý seznam nabízených termínů.
- **Zápis na vybraný termín a odepsání z vybraného termínu**
- **Upozornění na změnu obsazení termínu –** Umožní uživateli nastavit si upozornění pro případ, že se na vybraném zkouškovém termínu zaplní nebo uvolní místo. Pokud je pro daný termín nastavené upozornění a dojde ke změně obsazení tohoto termínu, zobrazí telefon standardní upozornění s informací o aktuálním stavu obsazení termínu. Student díky této funkci nebude muset kontrolovat, jestli se uvolnilo místo na jeho termínu, telefon jej upozorní automaticky.
- **Plánovač zkouškových termínů –** Nabídne studentovi možnost automatického naplánování zkoušek do co nejkratšího časového období. Uživatel bude moci nastavit omezující kritéria jako například počet volných dnů mezi zkouškami.

1.2.2 Uživatelské rozhraní

Aplikace bude rozdělena do tří hlavních částí:

1. **Seznam zkoušek a zapisování na zkoušky** – Tato sekce bude obsahovat seznam všech dostupných zkouškových termínů rozdělených podle předmětů. Po kliknutí na termín zobrazí detailní informace o vybraném termínu a umožní studentovi zapsat se termín, popřípadě se odepsat, pokud už je zapsán.
2. **Plánovač zkouškových termínů** – Bude obsahovat pole pro zadání omezujících kritérií uživatelem a jedno tlačítko pro potvrzení a naplánování termínů. Po kliknutí na tlačítko zobrazí v novém okně seřazený seznam termínů, které vyhovují zadaným kritériím.
3. **Upozornění** – Bude obsahovat seznam všech existujících upozornění a možnost jejich odstranění. Dále umožní uživateli nastavit nové upozornění pro vybraný termín.

Mimo výše uvedené hlavní sekce bude aplikace obsahovat také okno pro přihlášení uživatele a hlavní obrazovku s přehledem zapsaných termínů. Konkrétní rozmístění všech funkcí a jejich komponent bude záviset na operačním sytému, na kterém aplikace poběží, protože každý systém má odlišné standardní uživatelské rozhraní.

2 Operační systémy pro mobilní telefony

Na trhu je v dnešní době velké množství operačních systémů (OS), které jsou vyvíjené pro použití nejen v mobilních telefonech. Většina mobilních platforem podporuje kromě mobilních telefonů také další zařízení, jako například tablety nebo chytré hodinky.

2.1 Windows Phone

Windows Phone je mobilní operační systém vyvíjený společností Microsoft. Jeho předchůdcem je Windows Mobile, což je operační systém, který se dříve používal v PDA. Windows Mobile ale nebyl vhodný pro moderní dotyková zařízení a Microsoft proto vyvinul úplně nový systém pro dotykové telefony pod názvem Windows Phone. První verze tohoto OS nesla označení Windows Phone 7 a byla vydána v roce 2010 [3].

Druhá generace tohoto systému pod názvem Windows Phone 8 si kladla za cíl více propojit aplikace na telefonech a moderní aplikace na tabletech s Windows. Ve stejném duchu se nese i nejnovější verze tohoto OS, která vypouští slovo Phone a jmenuje se Windows 10 Mobile [3].

Obecně se dá říci, že platforma Windows Phone není úspěšná. Jedním z hlavních důvodů z pohledu uživatele je rozhodně nedostatek aplikací. Android i iOS mají ve svých oficiálních obchodech s aplikacemi více než 1,5 milionu aplikací. V obchodu Windows Phone Store je jich 340 000 [4]. Přestože to není malé množství aplikací, jejich kvalita je často nesrovnatelně nižší v porovnání s Androidem a iOS. Aplikace, které na Windows Phone Store najdeme, jsou často zastaralé a nedostávají včas potřebné aktualizace. Vývojáři se raději soustředí na lukrativnější Android a iOS. Oproti ostatním je u Windows výhoda v otázce bezpečnosti ve srovnání s Androidem.

2.1.1 Možnosti vývoje aplikací

Pro vývoj aplikací na Windows Phone 7 se nejčastěji používal Silverlight, což je vývojářský nástroj od společnosti Microsoft. V roce 2013 Microsoft zrušil jeho vývoj, a tak se pro vývoj na Windows Phone začaly používat jazyky C# nebo C++ [5].

Pro Windows Phone je také možné programovat za pomoci HTML5 a Javascriptu, což je velká výhoda pro vývojáře webových aplikací, kteří HTML5 a Javascript důvěrně znají [5].

2.1.2 Vývoj a publikace

Microsoft doporučuje pro vývoj aplikací používat Microsoft Visual Studio s vývojářskými nástroji pro Windows 10. Prostředí Visual Studio je placené, ale existuje i jeho neplacená varianta, kterou lze pro vývoj mobilních aplikací bez problémů použít. Všechny nástroje jsou ke stažení na webu Microsoftu [6].

Publikování aplikací na Windows Phone Store je zpoplatněno. Pro individuálního vývojáře účet stojí $19 USD, firemní účet stojí $99 USD. Jedná se o jednorázové poplatky, které se platí pouze jednou. Studenti mají vývojářský účet zdarma a mohou tak publikovat aplikace bez poplatků [7].

Před zveřejněním vytvořené aplikace musí nejprve dojít k její kontrole. Microsoft nabízí nástroj Windows App Certification Kit, který provede základní otestování aplikace. Po nahrání aplikace do Windows Phone Store musí ještě dojít k její certifikaci. Tento proces může trvat i několik dní a jeho délka závisí na složitosti aplikace [8].

Po certifikaci je aplikace vystavena na Windows Phone Store a uživatelé ji mohou stahovat.

2.1.3 Výhody a nevýhody

Výhody pro uživatele

- Známé uživatelské prostředí vizuálně podobné Windows na PC
- Velmi jednoduché ovládání
- Bezpečnost

Nevýhody pro uživatele

- Nedostatek aplikací
- Zastaralé a nekvalitní aplikace
- Velmi málo rozšířený systém

Výhody pro vývojáře

- Možnost programování v HTML a Javascriptu
- Kvalitní vývojářské nástroje od Microsoftu
- Nízký poplatek za vývojářský účet

Nevýhody pro vývojáře

- Malé zastoupení na trhu – aplikace nebudou tolik stahované

- Nekvalitní vyhledávání ve Windows Phone Store

2.2 iOS

Operační systém iOS vyvíjí společnost Apple pro svá mobilní zařízení iPhone, iPod a iPad. První verze iOS byla vydána společně s představením prvního telefonu iPhone v roce 2007 pod názvem iPhone OS 1. V roce 2010 nesla čtvrtá generace tohoto systému jméno iOS 4.

2.2.1 Možnosti vývoje aplikací

Aplikace pro iOS se od začátku mohly programovat v jazyku Objective C. V roce 2014 Apple představil Swift, programovací jazyk pro iOS a OS X, a dal tak vývojářům další možnost programování.

2.2.2 Vývoj a publikace

Pro vývoj aplikací na iOS je potřeba počítač s OS X. Apple nabízí vývojářům své vlastní vývojové prostředí pod názvem Xcode, které je ke stažení zdarma a obsahuje všechny potřebné nástroje pro vývoj iOS aplikací [9].

Obchod s aplikacemi pro iOS se jmenuje App Store. Pro publikování aplikací je potřeba mít vývojářský účet, který stojí $99 USD na rok. Apple nenabízí žádný přístup zdarma pro studenty [10].

Stejně jako u Windows Phone Store, i v App Store aplikace podléhá kontrole před jejím zveřejněním [11].

2.2.3 Výhody a nevýhody

Výhody pro uživatele

- Velké množství aplikací v App Store
- Aplikace jsou velmi kvalitní

Nevýhody pro uživatele

- Uzavřený systém
- Omezená možnost personalizace

Výhody pro vývojáře

- Vysoké zastoupení na trhu

- Relativně jednoduchá optimalizace aplikací z důvodu nízkého počtu fyzických zařízení

Nevýhody pro vývojáře

- Potřeba počítače s OS X, nelze programovat na Windows
- Přísná kontrola aplikací v App Store

2.3 Android

Operační systém Android je vyvíjen společností Google. Na rozdíl od Windows Phone a iOS se jedná o opensource OS. To znamená, že jej může kdokoli zcela zdarma používat a provádět v něm libovolné úpravy. Je to vidět na dnešních telefonech s Androidem, kde každé zařízení má trochu jiné uživatelské rozhraní a výrobci často přidávají i speciální funkcionalitu navrženou na míru každému zařízení. Je to bezesporu velká výhoda systému Android, ze které však plyne jeden problém.

Systém Android je poměrně často aktualizován. Nová verze Androidu vychází zpravidla každý rok a vývojáři aplikací i upravených OS by se měli snažit svůj software neustále aktualizovat. Spousta výrobců svá zařízení však nepodporuje dostatečně dlouhou dobu a jejich uživatelé tak nemají k dispozici poslední verzi pro OS Android.

První verze, Android 1.0, vyšla v roce 2008 [12]. Dnes je poslední vydaná verze Android 6.0 označovaná jako Android Marshmallow. Systém Android našel široké uplatnění a těší se velké popularitě. Kromě telefonů a tabletů funguje Android také na chytrých hodinkách, palubních deskách moderních automobilů, nebo v chytrých televizorech.

2.3.1 Možnosti vývoje aplikací

Většina aplikací pro Android se programuje v jazyce Java. K dispozici je také Android Native Development Kit (NDK), které dovoluje programátorovi implementovat části aplikace pomocí jazyků C a C++. Android NDK je doporučeno používat v případě, kdy aplikace vyžaduje vysoký výpočetní výkon nebo je graficky náročná [13].

2.3.2 Vývoj a publikace

Standardním nástrojem pro vývoj aplikací pro Android je vývojové prostředí Android Studio, které je ke stažení zdarma na stránkách Android Developers [14].

Pro publikování aplikací do obchodu s aplikacemi, který se u Androidu jmenuje Google Play, je potřeba zaplatit jednorázový poplatek $25 USD. Google nenabízí žádné slevy pro studenty.

2.3.3 Výhody a nevýhody

Výhody pro uživatele

- Velké množství aplikací v Google Play
- Možnost výběru z velkého množství fyzických zařízení
- Vysoká míra personalizace systému

Nevýhody pro uživatele

- Výrobci zařízení často neaktualizují software
- Relativně velké množství virů v porovnání s ostatními OS

Výhody pro vývojáře

- Vysoký podíl na trhu
- Možnost vytvoření vlastní úpravy systému
- Nízký poplatek za účet pro publikaci aplikací

Nevýhody pro vývojáře

- Nutnost optimalizace pro velký počet zařízení

2.4 Multiplatformní vývoj

V předcházejících odstavcích byly uvedeny doporučené možnosti programování pro jednotlivé systémy. Jednalo se však o nativní programování, tedy vývoj aplikací pouze pro danou platformu. Existuje velké množství frameworků, s jejichž pomocí lze psát hybridní aplikace, které budou fungovat na více systémech. Oblíbenými frameworky jsou například Sencha Touch [15], Ionic [16] nebo PhoneGap [17].

Uvedené frameworky (a většina ostatních) používají běžné technologie pro vývoj webových aplikací, jako HTML, CSS a Javascript. Je to výhoda pro mnoho vývojářů, kteří tyto jazyky ovládají. Další velkou výhodou těchto frameworků je, že stačí aplikaci napsat pouze jednou a bude fungovat na více OS.

Různé operační systémy mají různé designové požadavky na uživatelské rozhraní (UI). Frameworky z větší části pracují s vlastními komponentami UI a vypadají tak stejně na všech systémech, což nemusí být vždy žádoucí [18]. Z toho důvodu frameworky často

obsahují prvky UI, které jsou specifické pro podporované OS. Z toho vychází nutnost aktualizací takových prvků podle nových verzí operačních systémů. Nemusí se jednat pouze o UI komponenty, ale jakékoli API, které je k dispozici pro nativní vývoj. Pokud framework API nepodporuje, nelze ho využít. Je to jedna z nevýhod multiplatformních frameworků v porovnání s nativním vývojem. Nativní vývoj nám vždy umožňuje využívat nejnovější API a jednoduše tak aktualizovat a přizpůsobovat aplikaci posledním trendům v daném operačním systému.

Když Google v roce 2014 vydal Android verze 5.0 Lollipop, zároveň představil nový designový směr s názvem Material Design, který poměrně podstatně změnil vzhled Android aplikací [19]. Vývojáři nativních aplikací v té době použili oficiální nástroje a API, aby své aplikace aktualizovali a převedli na nový designový směr. Programátoři využívající některého z frameworků v takových případech musí počkat, než jejich framework bude nové API podporovat.

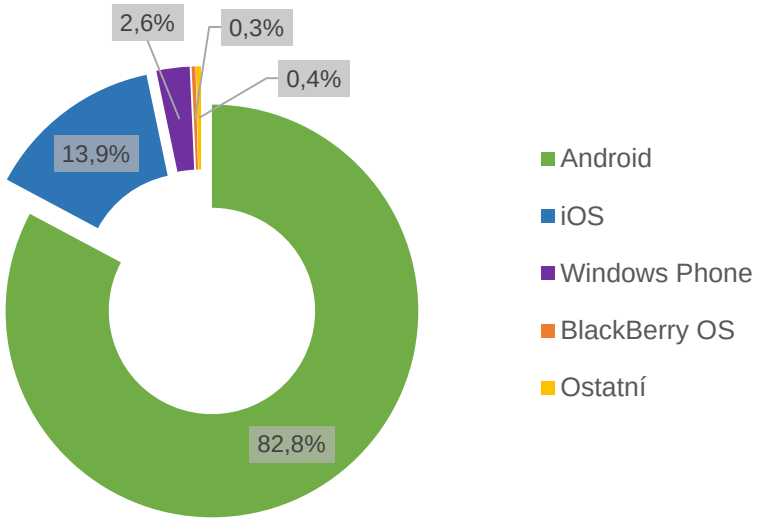
Existují i frameworky, které se umí pracovat s nativními API, například Xamarin. Pro tento framework se ale programuje v jazyce C#, čímž se ztrácí jednoduchost vývoje pomocí webových technologií [20].

Při rozhodování mezi nativním vývojem a frameworkem je vždy potřeba zvážit požadavky konkrétní aplikace.

2.5 Zhodnocení

Jednoznačně nejrozšířenějším OS pro mobilní telefony je v současnosti systém Android, který obsazuje 82,8% trhu. Druhým v pořadí je iOS s necelými 14% a Windows Phone na třetím místě má 2,6% trhu. Zbylé necelé procento patří ostatním systémům, které nejsou v současnosti příliš populární [21].

Obrázek 1: Rozšířenost OS pro mobilní telefony celosvětově (2015)



Zdroj: vlastní zpracování s použitím [21], 2016

Zmíněná čísla platí pouze pro celosvětový trh s mobilními telefony, nepočítají se sem tablety ani jiná zařízení. Při pohledu na statistiku z USA je patrná velká rozšířenost iOS. Na trhu USA má Android 52,1% podíl a iOS 43,5% [22].

2.5.1 Výběr OS

Hlavním kritériem při výběru byla rozšířenost systému a cenová dostupnost telefonů. Android má na trhu dominantní postavení a díky velkému množství zařízení jsou telefony s Androidem cenově dostupné i pro studenty. Další výhodou Androidu je dle autora této práce například možnost testování na fyzických zařízeních, kterých má k dispozici několik.

Z těchto důvodů bude aplikace Zkoušky vyvíjena nativně pro Android.

3 Nástroje vývoje aplikace pro Android

3.1 Vývojová prostředí

Vývojové prostředí je program, který pomáhá programátorovi při vývoji aplikace. Integruje všechny nástroje potřebné pro vývoj software pro danou platformu, aby byly programátorovi rychle přístupné [23]. Vývojovým prostředím se v angličtině říká Integrated Development Environments (IDE).

Pro pohodlný vývoj Android aplikací jsou potřeba Android SDK Tools. Je to soubor nástrojů a knihoven, které budeme potřebovat. SDK Tools lze stáhnout samostatně a programovat v podstatě v jakémkoli vývojovém prostředí. Zvláště v případě mobilních aplikací je však vhodné používat takové IDE, které je na takový vývoj připravené. Takové IDE totiž obsahuje velmi užitečné nástroje, jako například emulátory, debugger nebo návrhář uživatelského rozhraní.

3.1.1 Android Studio

Android Studio je IDE postavené na základech IntelliJ IDEA, což je populární a komplexní vývojové prostředí pro Javu [24]. Android Studio je v současné době oficiální nástroj pro vývoj aplikací pro Android a je ke stažení zdarma pro Windows, Linux a Mac OS X.

Toto IDE v sobě integruje všechny nástroje z Android SDK Tools. Nástroje jako SDK Manager nebo Android Virtual Devices Manager (AVD) je tak možné spustit pohodlně přímo z Android Studia a není třeba otevírat samostatné aplikace.

Android Studio dokáže výrazně urychlit práci s XML layouty. Při psaní XML kódu je možné si souběžně zobrazit preview, na kterém je možné vidět prováděné úpravy okamžitě, bez nutnosti spuštění aplikace. Kromě psaní XML kódu je možné se přepnout do režimu Design a použít grafické prostředí pro návrh layoutů. V tomto režimu lze také upravovat vlastnosti jednotlivých XML komponent. Vzhled komponent uživatelského rozhraní není ve všech verzích Androidu stejný a Android Studio umožňuje zobrazení navrženého UI tak, jak by vypadalo na vybrané verzi systému.

Android Studio obsahuje nástroje Image Asset Studio a Vector Asset Studio. Tyto nástroje umožňují programátorovi rychle importovat obrázky do projektu. Image Asset Studio generuje z vybraného obrázku ikony v několika rozlišeních a rozřadí je do

odpovídajících složek v systémové struktuře projektu. Vector Asset Studio pracuje na rozdíl od Image Asset Studia s vektorovými obrázky a převádí je do XML souboru.

Před vytvořením nové Java třídy v Android Studiu je možné si vybrat z několika šablon, které obsahují nejpoužívanější rozložení aktivit, fragmentů a služeb. Android Studio také automaticky registruje vytvořené komponenty do souboru AndroidManifest.xml.

3.1.2 Eclipse

Před vydáním Android Studia se pro vývoj aplikací používalo populární IDE Eclipse, pro které Google vydal plugin obsahující nástroje potřebné k vývoji Android aplikací. Plugin pod názvem Android Development Tools (ADT), obsahoval hlavně návrhář pro XML layouty a přidával do Eclipse odkazy na externí nástroje Android SDK Tools. V srpnu roku 2015 vyšla jeho poslední verze a byla ukončena jeho podpora [25].

3.1.3 Zhodnocení

U vývojových prostředí pro Android vlastně není z čeho vybírat. Jediným podporovaným IDE je Android Studio a proto si jej autor této práce vybral pro vývoj své aplikace.

3.2 Debugging a emulátory

Emulátor je virtuální stroj nahrazující fyzické zařízení, který slouží k rychlému a jednoduchému testování aplikací. Ideální je testovat aplikace přímo na zařízení, na kterém mají běžet. Android se však nachází na velkém množství zařízení, které se liší svou výbavou, rozlišením a velikostí displeje, typem procesoru nebo počtem kamer a senzorů. Testování aplikace na všech fyzických zařízení proto není prakticky možné. Emulátory umožňují vývojáři nastavit si verzi systému, typ zařízení, jeho displej, výbavu a další vlastnosti, a otestovat tak svou aplikaci na libovolném virtuálním zařízení.

Android SDK Tools obsahují emulátor, vývojáři však mohou použít jiný. Na trhu je poměrně velké množství emulátorů systému Android, většina z nich je ovšem zaměřená na hráče, kteří si chtějí užít Android hry na svém Windows zařízení, nikoliv na debugging aplikací.

3.2.1 Android Debug Bridge

Android Debug Bridge (ADB) je standardní nástroj v SDK Tools, který umožňuje komunikaci s připojenými zařízeními a emulátory [26].

ADB je možné rozdělit na tři komponenty - klient, server, a démon (daemon). Klient a server běží na počítači, na kterém probíhá vývoj aplikace, přičemž server je služba na pozadí. Daemon běží jako proces na pozadí připojeného zařízení nebo emulátoru. ADB server zajišťuje vlastní komunikaci mezi klientem a daemonem. Po jeho spuštění si server najde všechna připojená zařízení a emulátory a čeká na příkazy od klienta [26].

Příkladem komunikace pro debugging je nástroj logcat. Logcat je nástroj v SDK Tools, který sbírá všechny zprávy logovacího systému. Tyto zprávy pochází ze všech aplikací běžících na daném zařízení a nástroj logcat umožňuje zprávy filtrovat podle konkrétní aplikace, důležitosti zprávy, tagu a dalších vlastností. Logcat lze spustit pomocí ADB [27].

Při použití ADB nezáleží na tom, jestli se aplikace testuje na emulátoru nebo fyzickém zařízení.

3.2.2 Emulátor v Android SDK Tools

Součástí Android SDK Tools je emulátor, který umožňuje testování na každé verzi systému Android počínaje verzí 2.3.3 [28]. Nastavení a spuštění emulátoru je integrováno do Android Studia pomocí nástroje Android Virtual Device (AVD). AVD je nástroj z SDK Tools, který umožňuje vytvoření virtuálního zařízení a nastavení jeho vlastností.

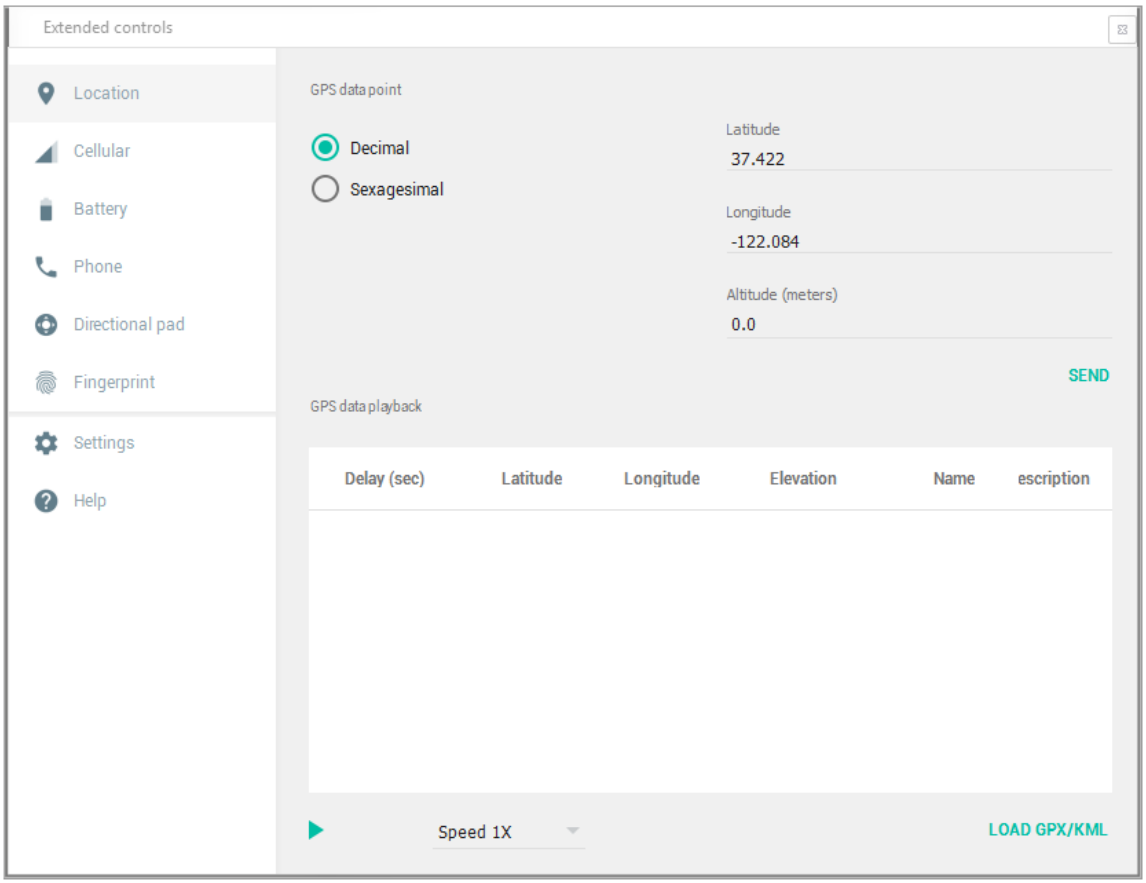
Mezi možnosti nastavení virtuálního zařízení patří:

- Verze systému
- Dostupný hardware – kamery, senzory, fyzická klávesnice
- Velikost operační paměti
- Emulovaná SD karta [29]

Po spuštění vlastního emulátoru je možné s ním virtuálně manipulovat. Základní panel funkcí obsahuje všechna tlačítka, která se na zařízení nachází, včetně ovládání hlasitosti. Dále nabízí možnost otočení zařízení o 90° a pořízení snímku obrazovky.

Zajímavější funkce se nacházejí v okně Extended controls. Na emulátoru je možné nastavit aktuální polohu telefonu pomocí GPS souřadnic, což je nezbytná funkce při vývoji aplikace, která pracuje s geolokačními daty. Lze také manipulovat se silou signálu a rychlostí a typem připojení k internetu. Tato funkce dokáže dobře napodobit stav, kdy se zařízení nachází na místě s pomalým či nespolehlivým připojením nebo se nachází v roamingu. Popsané stavy by se na reálném zařízení testovaly se značnými obtížemi.

Obrázek 2: Panel „Extended controls“ pro Android emulátor



Zdroj: vlastní zpracování, 2016

Mezi další funkce patří například možnost manipulace se stupněm nabití baterie. Dále je možné odeslat SMS zprávu na emulátor nebo vyvolat příchozí hovor. Za pomoci emulátoru je tak možné rychle otestovat a přizpůsobit vyvíjenou aplikaci velkému množství stavů, ve kterém se reálné zařízení může nacházet.

3.2.3 Genymotion

Genymotion je Android emulátor, který jako jeden z mála obsahuje vývojářské nástroje pro testování aplikací. Velkou výhodou tohoto emulátoru je jeho kompatibilita

s Android SDK Tools a Android Studiem [30]. Je tak možné využít všech vývojářských nástrojů a funkcí, které Android Studio nabízí. Genymotion může také fungovat zcela nezávisle na Android SDK Tools.

Genymotion je placený software, který pro jednotlivce stojí €99 na jeden rok. Existuje také verze dostupná zdarma, ta má však velmi omezené funkce [31]. Stejně jako standardní emulátor v Android SDK Tools nabízí možnost změny stavu baterie a nastavení GPS polohy. Ostatní funkce jsou přístupně pouze v placené verzi emulátoru.

Placená verze obsahuje další funkce, které ve standardním Android SDK Tools emulátoru neexistují. Genymotion například umožňuje manipulaci s některými prvky emulátoru přímo z Java kódu aplikace. Lze tak například nastavovat GPS polohu nebo stav baterie a automatizovat tak testování aplikace [30]. Další zajímavou funkcí je možnost použití vybraných senzorů z připojeného fyzického zařízení [30]. Vývojář může například spustit emulátor s libovolným nastavením, připojit fyzický telefon a používat jeho kameru nebo senzory pohybu a světla. Je tak možné skombinovat výhody virtuálních a fyzických zařízení pro účely testování.

3.2.4 Zhodnocení

Genymotion ve verzi zdarma nenabízí tolik funkcí a postrádá například možnost nastavení rychlosti a typu internetového připojení, což je velmi vhodný nástroj pro testování aplikací s přístupem k internetu. Autor práce se proto rozhodl používat pro vývoj své aplikace emulátor z Android SDK Tools.

4 Vývoj nativní aplikace pro Android

4.1 Struktura Android projektu

4.1.1 Adresářová struktura

Android aplikace se skládá z mnoha typů souborů, které obsahují zdrojové kódy, obrázky, ikony a další typy souborů. Tyto soubory jsou rozděleny do určité adresářové struktury, která musí být zachována. Podle této struktury se při exportu aplikace provede sestavení celé aplikace do jednoho souboru, která má příponu .apk. Android Studio po založení nového projektu automaticky vytvoří adresářovou strukturu a umisťuje soubory na správná místa.

Při programování nativních Android aplikací se běžně setkáme se dvěma typy zdrojových kódů – Java a XML. Obecně lze říci, že jazyk XML je použit na popis statických objektů, které se nemění. S pomocí XML se v Android projektu definují například uživatelská rozhraní. Za pomoci Javy se programuje samotná aplikační logika.

Důležité adresáře a jejich obsah:

- **app/build** – Obsahuje výstupní soubory po sestavení aplikace. Najdeme zde například používané externí knihovny, konfigurace a další automaticky generované soubory.
- **app/src/main/java** – Obsahuje všechny Java soubory dále rozdělené podle názvu balíčku, do kterého patří.
- **app/src/main/res** – Obsahuje všechny zdroje (resources), například XML soubory nebo ikony a obrázky [32].

Adresáře v app/src/main/res:

- **drawable** – Adresář obsahující obrázky. Může se jednat o bitmapové obrázky ve formátech PNG, JPEG a GIF nebo vektorové tvary uložené v XML souboru.
- **mipmap** – Adresář pro ukládání ikon aplikace.
- **layout** – Obsahuje návrhy uživatelského rozhraní (layouty) v XML souborech.
- **menu** – Obsahuje XML soubory popisující menu v aplikaci.
- **values** - Obsahuje XML soubory definující další zdroje (dále označované jako resources) jako barvy, řetězce nebo styly.

- **xml** – Obsahuje různé XML soubory, které nespadají do žádného adresáře [33].

4.1.2 Rozdělení a práce se zdroji (resources)

Adresáře ve složce /res mohou být dále rozdělené pro použití v konkrétní verzi systému nebo určitého rozlišení displeje. Mezi další možnosti rozdělení patří například orientace obrazovky, velikost displeje, dostupnost klávesnice nebo nastavení jazyka. Rozdělení týkající se vlastností displeje a obrazovky se často používají pro vytváření různých uživatelských rozhraní, které se například v závislosti na orientaci obrazovky mohou měnit. Použití rozdělení podle jazyka je možné použít například pro soubor strings.xml a jednoduše tak lokalizovat aplikaci pro několik různých jazyků [33].

Android definuje šest různých hustot rozlišení, pro které by vývojáři měli vytvářet své obrázky a ikony. Hustota rozlišení (density) je udávaná v jednotkách DPI (Dots per inch), která znamená počet pixelů na jeden čtverečný palec.

Tabulka 1: Hustoty rozlišení v Androidu

Zkratka	DPI
ldpi	120dpi
mdpi	160dpi
hdpi	240dpi
xhdpi	320dpi
xxhdpi	480dpi
xxxhdpi	640dpi

Zdroj: vlastní zpracování s použitím [34], 2016

Adresář, ve kterém se nacházejí obrázky pro rozlišení hdpi, by se jmenoval `drawable-hdpi`. Všechny obrázky a ikony, které aplikace používá, by měly být k dispozici ve všech uvedených rozlišeních. Android za běhu aplikace automaticky vybírá vhodné rozlišení obrázku podle zařízení, na kterém aplikace běží [34]. Při importování obrázků a ikon pomocí Android Studia se automaticky provede vygenerování všech potřebných rozlišení a jejich rozdělení do odpovídajících složek.

Podobným způsobem lze rozdělovat resources podle verze systému. Například ve složce s názvem `values-v21` najdeme XML resources pro verzi API 21 a vyšší. Rozdělení

podle verze se často používá například pro definování stylů, kdy různé verze systému používají různé knihovny.

Kdykoli aplikace potřebuje přistoupit k nějakému XML resource, pokusí se systém nalézt požadovaný soubor ve specifické složce. Pokud soubor ve složce neexistuje, bude použita resource z obecného adresáře. Například při přístupu k souboru `styles.xml` v adresáři `values` na Androidu API verze 21 se systém nejprve podívá do složky `values-v21` a bude hledat soubor `styles.xml`. Pokud jej nenalezne, použije `styles.xml` z obecné složky `values` [34].

4.1.3 Soubor R.java

Jak již bylo zmíněno výše, XML resources v Androidu se používají na definování podoby uživatelských rozhraní, stylů nebo například stringů. Uživatelská rozhraní obsahují komponenty, se kterými pracuje uživatel a je tedy nutné s nimi manipulovat. Soubor strings.xml může obsahovat názvy oken aplikace či popisy tlačítek v UI. Aby měli vývojáři přístup k těmto resources a mohli s nimi manipulovat přímo ve zdrojovém kódu v Javě, existuje ve struktuře Android projektu soubor `R.java`.

Soubor `R.java` je automaticky generovaný při každé kompilaci aplikace. `R.java` obsahuje třídy, ve kterých najdeme atributy typu integer. Jména tříd v `R.java` odkazují na typ objektu, se kterým chceme pracovat, a atributy uvnitř těchto tříd odkazují na konkrétní elementy v XML. Například přístup k elementu string s názvem `app_name`, který se nachází v souboru `strings.xml`, by vypadal takto – `R.string.app_name` [35] [36].

4.1.4 Oprávnění (permissions)

Z bezpečnostních důvodů nemají Android aplikace v základní konfiguraci oprávnění provádět takové operace, které by mohly nepříznivě ovlivnit jiné aplikace, systém nebo uživatele [37]. Mezi takové operace patří například přístup k internetu, používání kamer, přístup k SMS zprávám a jejich odesílání, nebo čtení a zápis uživatelských dat jako jsou kontakty, emaily, zprávy nebo události v kalendáři. Oprávnění lze rozdělit na dvě skupiny – normální a nebezpečné. Toto rozdělení je důležité pro udělování oprávnění za běhu aplikace [38]. Mezi normální oprávnění patří například povolení přístupu k internetu nebo ovládání bluetooth. Odesílání SMS a provádění hovorů patří do nebezpečných oprávnění.

Pokud aplikace bude provádět operaci, která vyžaduje oprávnění, musí uživatele požádat o schválení daného oprávnění. Uživatel tato oprávnění schvaluje ještě před instalací aplikace. Od verze Android 6.0 je možné žádat o oprávnění za běhu aplikace. V takovém případě nemusí uživatel přijímat všechna požadovaná oprávnění před samotnou instalací. V momentě, kdy aplikace potřebuje provést operaci, pro kterou je vyžadováno oprávnění, požádá uživatele o udělení daného oprávnění. Uživatel se pak může rozhodnout, jestli oprávnění udělí nebo neudělí. Za běhu musí aplikace žádat o udělení nebezpečných oprávnění, normální oprávnění lze schválit před instalací [38].

Pro vývojáře aplikací to znamená, že musí žádosti o oprávnění řešit v Java kódu. Uživatel se totiž může rozhodnout oprávnění neudělit a aplikace na to musí umět reagovat. Bez ohledu na verzi systému musí být všechna požadovaná oprávnění zapsána v `AndroidManifest.xml` [37].

4.1.5 Soubor AndroidManifest.xml

`AndroidManifest.xml` se nachází v kořenovém adresáři projektu a obsahuje základní informace o aplikaci, které Android potřebuje před jejím spuštěním [39]. V Android manifestu najdeme mimo jiné:

- Název Java balíčku, který bude sloužit jako unikátní identifikátor aplikace
- Manifest může obsahovat název i kódové označení verze aplikace
- Všechna oprávnění, která aplikace potřebuje
- Může také deklarovat oprávnění, které musejí mít jiné aplikace, které chtějí využívat komponent této aplikace (pokud aplikace takové komponenty má)
- Deklaruje minimální verzi API potřebnou pro spuštění aplikace
- Popisuje všechny komponenty, které aplikace obsahuje a jejich vlastnosti
- Obsahuje seznam externích knihoven, které aplikace používá [39]

4.2 Komponenty aplikace

Komponenty jsou základními stavebními bloky každé Android aplikace [40]. Všechny komponenty musejí být deklarovány v Android manifestu, aby systém věděl o jejich existenci a mohl je používat. Každá z těchto komponent může představovat vstupní bod do aplikace z pohledu systému Android, pouze některé jsou vstupním bodem pro uživatele [40].

Existují čtyři typy aplikačních komponent:

- **Aktivity (Activities)** – Aktivita představuje jednu obrazovku s uživatelským rozhraním.
- **Služby (Services)** běží na pozadí a jsou používány k provádění operací, které mohou trvat delší dobu.
- **Poskytovatelé obsahu (Content providers)** – Slouží k jednoduchému přístupu k datům, které aplikace sdílí s jinými aplikacemi.
- **Přijímače (Broadcast receivers)** – Komponenta, která přijímá a reaguje na rozeslané broadcasty.

Android umožňuje všem aplikacím spustit komponentu jiné aplikace. Komponenta druhé aplikace musí mít povolené spouštění z jiných aplikací a obsahovat obecně užitečnou funkcionalitu. Znamená to, že vývojáři mohou použít obecné funkce aplikací, které uživatel má už nainstalované.

V praxi se takto využívají hlavně aplikace, které jsou na každém zařízení nainstalované z výroby. Například, pokud potřebuje naše aplikace odeslat zprávu s obrázkem, nemusí obsahovat komponentu, která pořídí fotografii. Aplikace může zavolat komponentu jiné aplikace, která fotografii pořídí a výsledný obrázek vrátí zpět do naší aplikace. V zařízení může existovat více aplikací, které takovou funkcionalitu umožňují. V takovém případě si uživatel může vybrat, kterou z aplikací chce použít.

Jak již bylo řečeno, každá komponenta může sloužit jako vstupní bod do aplikace. V praxi to znamená, že každá aplikace má několik vstupních bodů, nikoliv pouze jeden. Neexistuje zde žádná hlavní spouštěcí metoda, jako například `main`, která je známá z Javy [40].

4.3 Intent (záměry)

V předcházející kapitole bylo zmíněno, že každá Android aplikace může spouštět komponenty jiné aplikace. Přestože toho vývojáři mohou docílit pouze několika řádkami kódu, z bezpečnostních důvodů nedochází ke komunikaci mezi dvěma aplikacemi a není tak možné, aby aplikace přímo spustila komponentu jiné aplikace [40].

Pro spuštění komponenty, ať už se jedná o vnitřní komponentu nebo komponentu jiné aplikace, se používá intent. `Intent` je objekt, který zajišťuje komunikaci mezi komponentami. Pomocí intentu můžeme spustit aktivitu, službu a rozeslat broadcast.

Content providers se pomocí intentů nespouštějí. Do objektu typu `Intent` je také možné vložit data, která chceme přenést do spouštěné komponenty [41].

Pro spuštění aktivity slouží metoda `startActivity`, jejímž argumentem je právě objekt typu `Intent`. Takto může vypadat kód pro spuštění aktivity s názvem `HomeActivity`, do které přenášíme data ve formě ID čísla [41].

```
Intent intent = new Intent(this, HomeActivity.class);
intent.putExtra("id_cislo", 12);
startActivity(intent);
```

`HomeActivity.class` je třída, ve které se aktivita `HomeActivity` nachází. Pro přidání ID čísla do intentu se používá metoda `putExtra`, jejíž první parametr je klíč, pomocí kterého se k hodnotě bude přistupovat a druhým parametrem je samotná hodnota. Obdobným způsobem lze pomocí metody `startService` spustit službu.

Intenty lze rozdělit na implicitní a explicitní:

- **Explicitní** intenty mají konkrétně specifikovaný název komponenty. Tento přístup tedy lze použít pouze v případě, kdy známe celý název třídy, ve kterém se komponenta nachází. Typicky se jedná o komponenty uvnitř aplikace. V příkladu, který je uvedený výše, je použit explicitní intent.
- **Implicitní** intenty deklarují obecnou akci, kterou by komponenta měla provést, ale nespecifikují její název. Tento typ intentu se používá v případě, kdy aplikace spouští komponentu jiné aplikace. Příkladem může být otevření webové stránky. Stačí vytvořit intent s akcí pro otevření webové stránky. Systém vyhledá aplikace, které umějí otevřít webovou stránku, a dá uživateli na výběr, kterou z dostupných aplikací použít [41].

Implicitní intent pro otevření webové stránky může vypadat například takto:

```
Intent intent = new Intent();
intent.setAction(Intent.ACTION_VIEW);
intent.setData(Uri.parse("http://google.com"));
startActivity(intent);
```

Akce intentu se nastavuje pomocí metody `setAction`. V tomto příkladu používáme akci s názvem `ACTION_VIEW`, což je obecná akce pro zobrazení dat přiložených v intentu. Neslouží tedy pouze pro zobrazení webové stránky. Data do intentu přidáme metodou `setData`, jejíž parametr je objekt typu `Uri`. Po zavolání metody `startActivity` vyhledává systém pouze takové aplikace, které jsou schopné

zpracovat daný typ intentu i data, která jsou v něm přiložená. V tomto příkladu systém vyhledá webové prohlížeče, protože v datech je webová stránka [42].

Tento typ akce lze však kombinovat i s jiným typem dat. Například pro zobrazení seznamu kontaktů lze použít následující data:

```
intent.setData(Uri.parse("content://contacts/people/"));
```

Některé akce vyžadují, aby komponenta vrátila výsledná data zpět do aplikace. Takovou akcí je například už uvedený příklad, kdy aplikace potřebuje pořídit fotografii. Je pro to potřeba vytvořit implicitní intent podobně jako v příkladech výše. Na rozdíl od těchto příkladů však potřebujeme pořízenou fotografii vrátit zpět do aplikace. Pro tyto případy existuje metoda `startActivityForResult`, pomocí níž lze spustit aktivitu, která vrátí výsledná data. Tato ukázka kódu vyvolá spuštění aplikace, která dokáže pořídit fotografii:

```
final int REQUEST_IMAGE_CAPTURE = 1;
Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
if (takePictureIntent.resolveActivity(getPackageManager()) !=
null) {
    startActivityForResult(intent, REQUEST_IMAGE_CAPTURE);
}
[43]
```

Vytvoření intentu probíhá stejným způsoben jako v předcházejících příkladech, pouze akce je zadána přímo v konstruktoru. Argument `REQUEST_IMAGE_CAPTURE` je libovolný kód, který slouží pro identifikaci požadavku a bude použit pro získání pořízené fotografie. Před voláním metody `startActivityForResult` je ještě provedena kontrola za pomoci metody `resolveActivity`. Ta vrací první nalezenou komponentu, která dokáže provést požadovanou akci. Pokud by taková aplikace neexistovala, došlo by ukončení naší aplikace. Z toho důvodu by měla být tato kontrola vždy prováděna [43].

Pro získání pořízené fotografie je potřeba překrýt metodu `onActivityResult`. Překrytí této metody a získání dat vypadá takto:

```
@Override
protected void onActivityResult(int requestCode, int resultCode,
Intent data) {
    if (requestCode == REQUEST_IMAGE_CAPTURE && resultCode ==
RESULT_OK) {
        Bundle extras = data.getExtras();
        Bitmap imageBitmap = (Bitmap) extras.get("data");
    }
}
[43]
```

V `onActivityResult` se nejprve kontroluje, že se skutečně jedná o data vrácená z našeho požadavku o pořízení fotografie. Zde se použije kód, který byl zvolen v `startActivityForResult`. Parametr `resultCode` je roven `RESULT_OK` v případě, že byla operace provedena správně. Pokud by byla operace přerušena, byl by `RESULT_CANCELED`. Další kódy jsou definované ve třídě `Activity`. Pořízenou fotografii nalezneme v intentu s názvem `data`. Konkrétní název klíče, pod kterým najdeme požadovaná data, záleží na komponentě, která byla použita. Teoreticky může každá kamerová aplikace vracet pořízené obrázky odlišným způsobem. Uvedený způsob je používaný ve výchozí aplikaci fotoaparátu v Androidu [43].

V předcházejících odstavcích byly použity dva způsoby pro předání dat do intentu.

- **Metoda `setData`** se používá pro nastavení dat implicitního intentu. Tato data bude používat cílová komponenta pro dokončení požadované akce. Podle těchto dat také systém Android vybírá aplikace, které jsou vhodné pro danou akci.
- **Metoda `putExtra`** se používá pro vložení libovolných dat do intentu ve formě key-value-pair. V případě implicitního intentu se při vybírání vhodných komponent pro provedení akce na tato data vůbec nehledí.

Vedle metody `putExtra` je také možné použít `putExtras`. Parametrem `putExtras` je `Bundle`. `Bundle` je objekt, do kterého lze vkládat hodnoty a jejich klíče. Key-value-pairs je možné ukládat do objektu `Bundle` i přímo do intentu. Výhodou `Bundle` je seskupení všech dat do jednoho objektu, což je vhodné v případě, kdy potřebujeme stejná data předat to dalšího intentu.

Odesílání dat v intentu bylo popsáno v předcházejících odstavcích, zde je ukázka kódu, pomocí kterého lze získat přenášená data v cílové aktivitě [44].

```
Intent intent = getIntent();
int cislo = intent.getIntExtra("id_cislo", 0);
Bundle data = intent.getExtras();
```

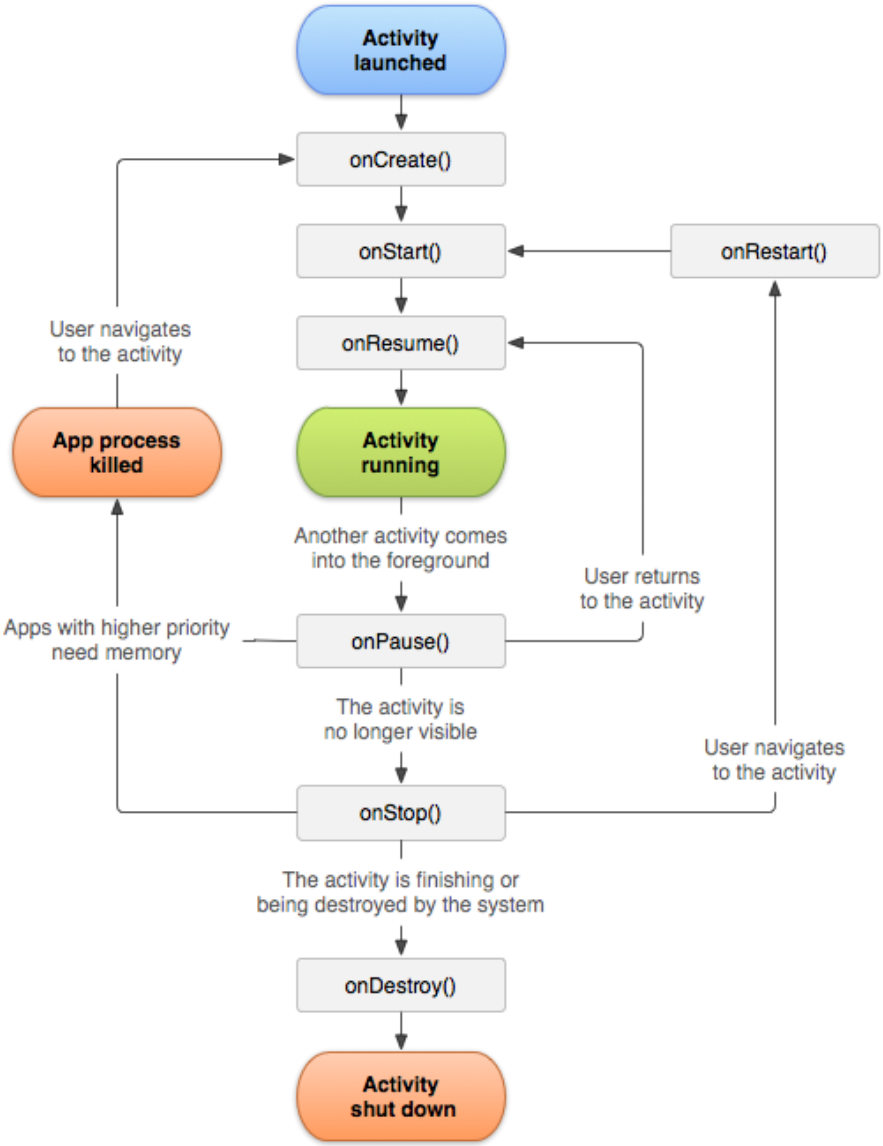
Metoda `getIntent` vrací intent, který spustil danou aktivitu. V tomto intentu nalezneme data v takové podobě, v jaké byla uložena [44]. Metoda `getExtras` vrátí objekt `Bundle` uložený v intentu. K jednotlivým key-value-pairs v intentu se přistupuje pomocí metod, které odpovídají typu vybraného parametru. V příkladu je uvedena metoda pro získání hodnoty typu `int`. První parametr této metody je klíč hodnoty, druhým parametrem je výchozí hodnota, která bude vrácena v případě, že hodnota se zadaným klíčem v intentu neexistuje.

4.4 Aktivity

Aktivita je aplikační komponenta, která poskytuje obrazovku, se kterou může uživatel pracovat. Každá aktivita je samostatným oknem, na které lze vykreslit uživatelské rozhraní. Aktivita běžně zabírá celou obrazovku, může však být i menší a částečně překrývat okna pod ní [45].

Řízení životního cyklu aktivity se provádí překrytím callback metod. Callback metody jsou takové metody, které jsou volané systémem po dokončení určité operace.

Obrázek 3: Životní cyklus aktivity



Zdroj: [45], 2016

První metodou, která je volána po spuštění aktivity, je `onCreate`. Překrytí této metody je povinné. V `onCreate` je také nutné volat metodu `setContentView`, pomocí které se provede nastavení layoutu uživatelského rozhraní [45]. V metodě `onCreate` by se měly provádět takové operace, které je třeba provést pouze jednou po spuštění aktivity. Jedná se hlavně o získání dat ze spouštěcího intentu nebo inicializaci proměnných.

Po `onCreate` následuje metoda `onStart`. Z diagramu je patrné, že se jedná o metodu, která je volána také po restartu aktivity. Proto by se zde měly provádět takové

operace, které jsou nutné pro každé zobrazení aktivity. Příkladem může být načtení aktuálních dat z databáze a jejich zobrazení [45].

`OnResume` je poslední metodou, která je volána před tím, než je aktivita skutečně spuštěna. Následuje po `onStart`, nebo po `onPause`. V `onResume` by se měly provádět jen ty nejnutnější operace, protože je to nejčastěji volaná metoda z trojice `onCreate`, `onStart` a `onResume` [45].

Po průchodu všemi třemi zmíněnými metodami je aktivita skutečně spuštěna a uživatel ji může používat. Přerušení běhu aktivity může nastat několika způsoby a prochází další sérií callback metod [45].

`OnPause` je první metodou, která je volána, když je aktivita částečně překryta jinou aktivitou. Nejčastěji se jedná o překrytí dialogem nebo upozorněním. Aktivita v tu chvíli není v popředí, ale je alespoň částečně stále viditelná. Aktivita v tomto stavu je pozastavená (paused), objekt aktivity je však stále uložen v paměti. V této metodě by mělo docházet k uložení kritických dat a ukončení operací, které by mohly zbytečně vytěžovat procesor, jako například animace. Pokud v tomto bodě dojde k návratu do původní aktivity, bude zavolána metoda `onResume`. Aktivita bude zobrazena tak, jak ji uživatel opustil, protože byla uložena v paměti [45].

Důležitý je fakt, že aplikace může být po průchodu metodou `onPause` ukončena systémem z důvodu nedostatku paměti. V případě pozastavené aktivity je ukončení systémem velmi nepravděpodobné, protože standardně před ukončením dochází k volání `onStop` a `onDestroy`. Pokud je to však nutné, může systém ukončit pozastavenou aktivitu i bez volání těchto metod. Je proto důležité zahájit operaci ukládání kritických dat, jinak by mohlo dojít k jejich ztrátě [45].

Pokud je aktivita plně překryta jinou aktivitou, dochází k volání metody `onStop` a aktivita je zastavena (stopped). Stále však zůstává uložena v paměti. V případě, že se uživatel vrátí do aktivity, která je zastavena, dojde k volání metod `onRestart` a následně `onStart` [45].

`OnRestart` je metoda volaná při restartu aktivity před metodou `onStart`. Často není vůbec nutné `onRestart` implementovat, pokud je však potřeba měnit implementaci metody `onStart` podle toho, jestli byla aktivita právě vytvořena nebo se restartuje, je vhodným řešením právě využití této metody [45].

Před řádným ukončením aktivity dochází k volání metody `onDestroy`. Aktivita může být ukončená systémem, nebo samotnou aplikací voláním metody `finish`. Pro rozeznání těchto dvou typů ukončení lze použít metodu `isFinishing` [45].

Technicky by mohlo v extrémních případech dojít k ukončení aplikace, která se nachází v jakémkoli stavu. Jedná se však pouze o extrémní případy, kdy systém nutně potřebuje uvolnit paměť. Z toho důvodu bychom měli provádět operace jako ukládání uživatelských dat v metodách `onPause` a `onStop` [45].

V přechozích odstavcích bylo uvedeno, že aktivita, která je pozastavená nebo zastavená, zůstává uložena v paměti. Stav této aktivity tedy zůstává nezměněn, a pokud se do ní uživatel vrátí, bude zobrazena tak, jak ji uživatel opustil. V případě jednoduché aktivity, která neukládá uživatelská data, proto často není potřeba implementovat metody `onStop`, `onRestart` a `onStart` [45].

Kromě uživatelských dat je pro účely zobrazení aktivity potřeba ukládat také stavová data o aktivitě. Může se jednat například o pozice a stav prvků uživatelského rozhraní. Pokud je v aktivitě rozbalovací menu, uživatel očekává, že po návratu do aktivity budou v menu rozbalené stejné položky, jako při opouštění aktivity. Jak bylo popsáno výše, aktivita zůstává uložena v paměti, dokud nedojde k jejímu ukončení. Existují však případy, kdy je potřeba ponechat stavová data aktivity i po tom, co byla aktivita zcela ukončena. Například při změně orientace obrazovky dochází k ukončení a opětovnému spuštění aktivity. Pro tyto případy budeme rozhodně chtít ukládat stavová data a zobrazovat uživateli aktivitu v takovém stavu, v jakém ji opustil [45].

Pro ukládání a načítání stavových dat slouží metody `onSaveInstanceState` a `onRestoreInstanceState`. Druhá z uvedených metod slouží pro získání uložených dat. Stavová data jsou přenášena v objektu typu `Bundle`. `Bundle` se stavovými daty je dostupný jak v metodě `onRestoreInstanceState`, tak v metodě `onCreate`. `OnRestoreInstanceState` je volána po `onStart`, před `onResume`, ovšem pouze v případě, že je aktivita právě vytvářena. Znamená to, že `onRestoreInstanceState` není volána při návratu do aktivity, která byla zastavena, a proto fakticky nezáleží na tom, jestli stavová data získáme z metody `onCreate` nebo `onRestoreInstanceState`. Druhou uvedenou je však doporučeno používat z důvodu možnosti změny implementace při dědění aktivity [45].

`OnSaveInstanceState` slouží pro ukládání stavových dat. Může být volána před `onPause`, `onStop` nebo také nemusí být volána vůbec. Neurčitá pozice této metody v životním cyklu aktivity je způsobená faktem, že se systém může rozhodnout ukončit aktivitu za účelem uvolnění paměti ve stavu zastaveno i pozastaveno. Stavová data aktivity jsou použita pouze v případě, kdy došlo k nucenému ukončení aktivity systémem. Z toho vyplývá, že existují případy, kdy není vůbec nutné stavová data ukládat, protože určitě nebudou použita. Jedná se o ukončení aplikace uživatelem, nebo ukončení aktivity metodou `finish`. `OnSaveInstanceState` automaticky ukládá data všech prvků uživatelského rozhraní v aktivitě. Například zadaný text nebo zaškrtnutý checkbox bude automaticky uložen a po obnovení aktivity znovu automaticky nastaven [45].

4.5 Fragmenty

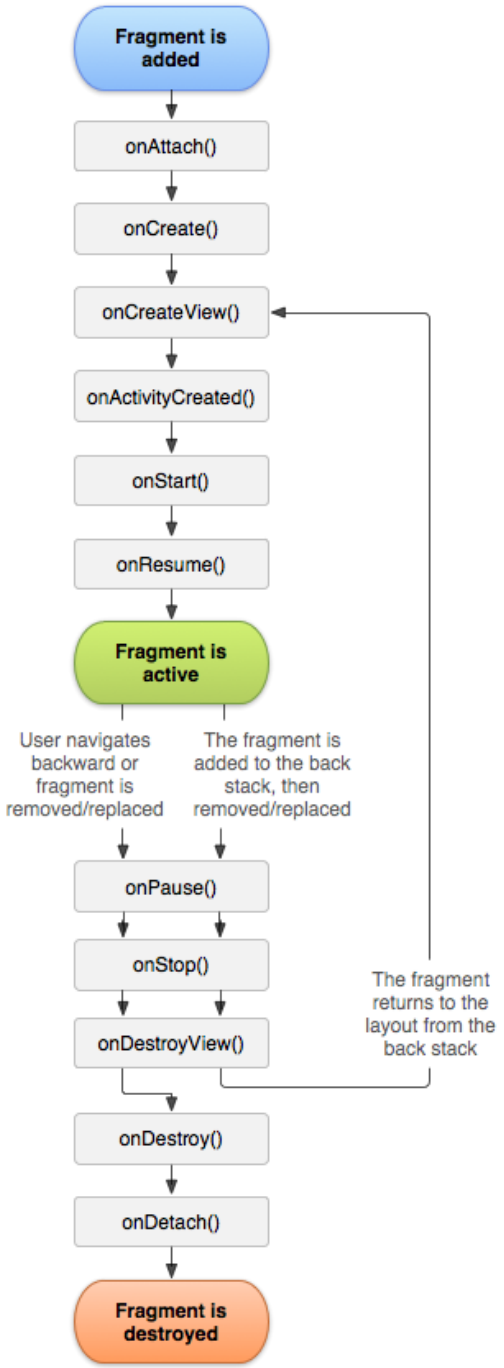
Fragmenty představují části obrazovky s uživatelským rozhraním. Poprvé byly představeny ve verzi Android 3.0, kde sloužily hlavně pro tvorbu flexibilních UI pro tablety. Protože tablety mají mnohem větší obrazovky než telefony, vejde se na jejich displej více informací. Například aplikace, která funguje jako emailový klient, bude na telefonu obsahovat aktivitu se seznamem emailů a po výběru konkrétního emailu spustí novou aktivitu, ve které se zobrazí vybraný email. Na dostatečně velkém displeji lze mít vše na jedné obrazovce, tedy v jedné aktivitě. Bez použití fragmentů by pro optimalizaci aplikace pro větší displeje bylo nutné vytvořit novou aktivitu, která by v sobě kombinovala funkcionalitu aktivity pro zobrazení detailu emailu a aktivity se seznamem emailů [46].

Fragmenty tento problém řeší díky těmto vlastnostem:

- Na jednu obrazovku lze vložit více fragmentů
- Mají životní cyklus podobný aktivitám, který lze stejným způsobem řídit

Každý fragment je součástí nějaké aktivity. V aktivitě je možné fragmenty vytvářet, přidávat je na určitá místa v UI a dále s nimi manipulovat. Protože fragment žije uvnitř aktivity, je také závislý na jejím životním cyklu. Samotný životní cyklus fragmentu proto obsahuje více metod, než cyklus aktivity [46].

Obrázek 4: Životní cyklus fragmentu



Zdroj: [46], 2016

Callback metody `onCreate`, `onStart`, `onResume`, `onPause`, `onStop` a `onDestroy`, které byly popsané v životním cyklu aktivity, se chovají stejně i u fragmentů. Mezi metody specifické pro fragment patří:

- **`OnAttach`** – Je volána po přiřazení fragmentu k aktivitě.

- **`OnCreateView`** – Zde by mělo dojít k vytvoření prvků uživatelského rozhraní a objektu View, který obsahuje všechny tyto prvky. Tento View objekt je návratovým typem onCreateView. Pokud by tato metoda vracela null, fragment nebude mít žádné UI.
- **`OnActivityCreated`** – Tato metoda ulehčuje řízení fragmentu v závislosti na stavu aktivity, do které patří. Je volána po tom, kdy aktivita dokončila svou metodu onCreate.
- **`OnDestroyView`** – Je volána když dochází k odstranění prvků UI patřících do daného fragmentu.
- **`OnDetach`** – Je volána po odpojení fragmentu od aktivity [46].

Stejně jako aktivita, i fragment se může nacházet ve stavu pozastaven nebo zastaven. Související callback metody jsou často volány právě v závislosti na stavu aktivity. Například při volání metody `onPause` v aktivitě bude volána i `onPause` ve fragmentu. Pokud je překryta aktivita, je překrytý i fragment, který do ní patří.

To však nemusí platit vždy. Fragmenty mohou mít například formu dialogu a částečně tak překrývat svou aktivitu. V takovém případě nebude volána `onPause`, protože aktivita není překrývána jinou aktivitou, ale vlastním fragmentem. Přestože se tedy životní cyklus každého fragmentu řídí samostatně, je potřeba si uvědomit, že je závislý na aktivitě, do které patří, a naopak životní cyklus aktivity není ovlivňován jejími fragmenty, ale pouze ostatními aktivitami tak, jak bylo popsáno v předcházející kapitole.

4.5.1 Vložení fragmentu do aktivity

Fragmenty lze stejně jako prvky UI deklarovat v XML kódu, nebo je vytvořit programově v Javě.

Pro deklaraci v XML existuje element fragment. Zde je ukázka XML kódu s tímto elementem:

```
<fragment android:name="eu.polivkal.zkouskyv1.PrehledFragment"
    android:id="@+id/fragment_prehled"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

V atributu `android:name` je specifikován název třídy, ve které se fragment nachází, včetně balíčku. Aby bylo možné s fragmentem dále manipulovat, je nutné mu přiřadit id v atributu `android:id`, nebo unikátní string v atributu `android:tag` [46].

Pro získání instance objektu fragmentu, který byl deklarován v XML, slouží metody `findFragmentById` a `findFragmentByTag`, které jsou součástí třídy `FragmentManager`. Instanci požadovaného fragmentu lze získat následujícím způsobem:

```
PrehledFragment prehledFragment = (PrehledFragment)
getFragmentManager().findFragmentById(R.id.fragment_prehled);
```

Častým způsobem pro přidávání fragmentů je programové vytváření v Java kódu. Při použití tohoto konceptu se většinou vytvoří prázdný kontejner deklarovaný v XML layoutu aktivity, do kterého se později vloží potřebný fragment. Takových kontejnerů lze v layoutu vytvořit několik a později do nich vkládat libovolné fragmenty.

```
<FrameLayout
    android:id="@+id/content_frame"
    android:layout_width="match_parent"
    android:layout_height="match_parent"></FrameLayout>
```

Uvedený kontejner má podobu FrameLayoutu, je však možné vybrat i jiný layout element, protože později bude nahrazen nějakým fragmentem. Zde je ukázka jednoduchého kódu, který zajistí vytvoření fragmentu s názvem `PrehledFragment` a jeho vložení do kontejneru s id `content_frame`.

```
Fragment fragment = PrehledFragment.newInstance();
getFragmentManager().beginTransaction().replace(R.id.content_fra
me, fragment).commit();
```

Uvedený kód vytváří novou transakci, ve které provede záměnu elementu s id `content_frame` za vytvořený fragment. Transakce umožňuje objekt `FragmentTransaction`, který kromě uvedené metody `replace` obsahuje také metody `add` a `remove`. Metoda `replace` provede záměnu elementu za daný fragment a metody `add` a `remove` zajišťují přidání a odebrání fragmentů. Všechny provedené transakce lze vkládat do zásobníku (back stack), který je řízen aktivitou a umožňuje uživateli vrátit se do předchozího stavu fragmentu pomocí tlačítka Zpět [46].

4.6 Služby

Služba je aplikační komponenta, která není viditelná uživateli, běží na pozadí a provádí operace, které mohou trvat delší dobu. Příkladem takové operace je například přehrávání hudby, komunikace se serverem nebo zápis a čtení dat [47].

Služby mohou být dvojího typu:

- **Spuštěné (started**) služby jsou spuštěny pomocí metody `startService` v aktivitě. Spuštěné služby mohou běžet na pozadí, i když došlo k ukončení komponenty, která službu spustila, nebo když uživatel opustí (ale neukončí) danou aplikaci. Takové služby většinou provádí jeden typ operace, po jejíž skončení se sami ukončí [47].
- **Napojené (bound)** služby jsou spouštěny metodou `bindService`. Tyto služby umožnují aplikačním komponentám po spojení se službou vyžádat provedení nějaké operace a vrácení jejího výsledku. Příkladem takové operace je například dotaz na databázi a vrácení výsledku. Napojené služby mohou obsluhovat více komponent současně. Služba může být klasickou spuštěnou službou a zároveň umožňovat napojování [47].

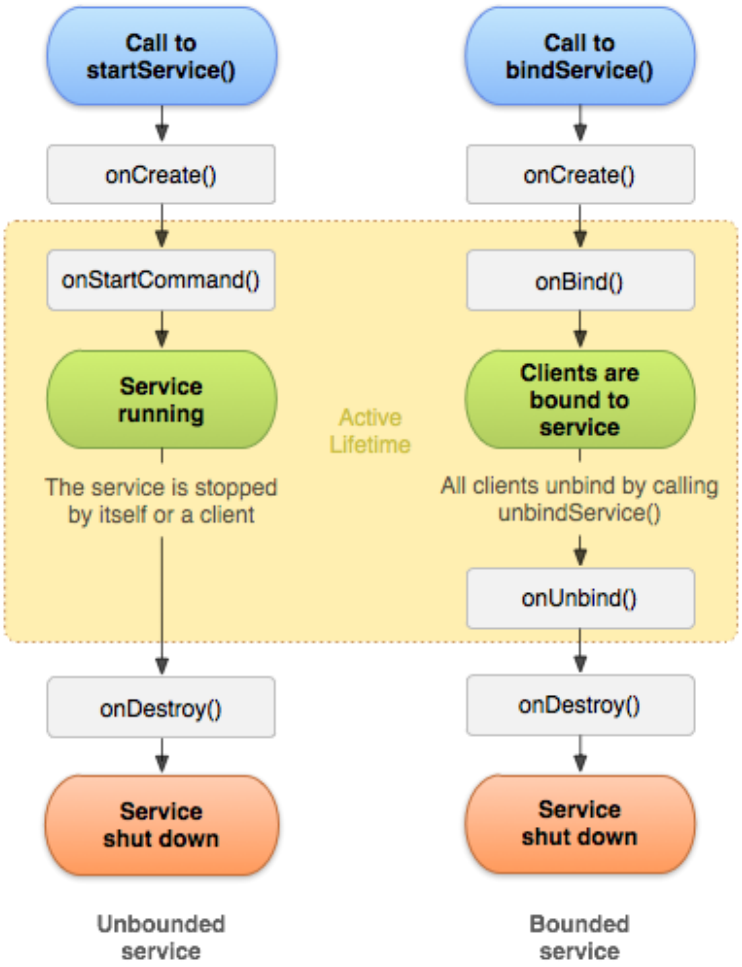
Přestože bylo zmíněno, že služba běží na pozadí, neznamená to, že je na jiném vlákně než zbytek aplikace. Služba běží na stejném vlákně a z toho důvodu je potřeba přesunout některé operace, jako například práce se sítí, na jiné vlákno. Toto téma je blíže rozebráno v následující kapitole.

4.6.1 Životní cyklus služby

Služby obsahují již známé callback metody `onCreate` a `onDestroy`. Jejich životní cyklus se liší podle toho, jakým způsobem byly volány, respektive o jaký typ služby se jedná.

Při spuštění služby metodou `startService` je volána metoda `onStartCommand`. Pokud služba nebude spouštěna pomocí `startService`, není nutné `onStartCommand` implementovat. Ukončení služby je možné provést metodami `stopSelf` nebo `stopService` [47].

Obrázek 5: Životní cyklus služby



Zdroj: [47], 2016

V případě napojení služby na jinou aplikační komponentu je volána metoda `onBind`. Ta musí vracet rozhraní `IBinder`, které budou komponenty používat pro komunikaci se službou. `OnBind` musí být implementována vždy, pokud však aplikace nebude využívat možnosti napojení služby na komponentu, je vhodné vracet v této metodě `null` [47].

Poslední callback metodou je `onUnbind`, která je volána po odpojení všech komponent od služby [48].

4.7 Procesy a vlákna

Při prvním spuštění aplikace dojde k vytvoření jednoho procesu a vlákna, ve kterém se budou spouštět všechny komponenty dané aplikace. V manifestu lze toto nastavení

změnit a přiřadit vybraným komponentám jiný proces. Slouží k tomu atribut `android:process` v XML elementu dané komponenty [49].

Vlákno, které se vytváří při spuštění aplikace, se nazývá hlavní (main thread), nebo také UI vlákno. Protože na něm běží všechny komponenty aplikace a tedy i aktivity, dochází na tomto vlákně i ke zpracování UI prvků a všech interakcí uživatele s UI. Pokud by se na tomto vlákně měly spouštět dlouhotrvající operace jako například stahování a nahrávání souborů, nebylo by možné po dobu trvání dané operace aplikaci používat. Pokud by na hlavním vlákně běžela operace blokující UI po dobu delší než 5 sekund, systém zobrazí dialog s upozorněním „Aplikace neodpovídá“. Z toho důvodu by se vývojáři měli snažit dodržet dvě jednoduchá pravidla pro práci s UI vláknem:

1. Neblokovat UI vlákno, tj. nespouštět v něm žádné dlouhotrvající operace nebo operace, které mohou být provedené jinde.
2. Přistupovat k UI prvkům pouze z UI vlákna [49].

4.7.1 Vytvoření nového vlákna

V Androidu lze vytvořit nové vlákno například pomocí tříd `Thread` nebo `Runnable`, které jsou používány v Javě. Problém však nastává v případě, kdy je po dokončení operace potřeba manipulovat s prvky UI. Přístup k UI z jiného než hlavního vlákna totiž porušuje výše uvedené druhé pravidlo. Při pokusu o manipulaci s UI prvky z jiného vlákna se aplikace může chovat nepředvídatelně a ve většině případů dojde k jejímu ukončení [49].

4.7.2 AsyncTask

Android nabízí možnosti, jakými tento problém vyřešit za použití zmíněných tříd, ale při větším množství kódu se stávají nepřehlednými a komplikovanými. Nejlepším řešením je použití třídy `AsyncTask`.

`AsyncTask` zjednodušuje provádění operací mimo hlavní vlákno a následnou projekci výsledků do UI [50]. Pro použití AsyncTasku je třeba vytvořit třídu, která dědí od třídy `AsyncTask`.

```
public class MyNetworkTask extends AsyncTask<Void, Integer,
String>
```

AsyncTask je definován třemi generickými datovými typy:

1. **Params** – Typ parametru, které chceme AsyncTasku předat.

2. **Progress** – Pokud bude AsyncTask zobrazovat svůj postup, například 0% - 100% při stahování souboru, jedná se o datový typ, který bude použit pro zobrazení postupu. Většinou se bude jednat o číslo a bude tedy použit Integer.
3. **Result** – Datový typ výsledku, který bude AsyncTask vracet [50].

AsyncTask obsahuje čtyři callback metody, pomocí kterých je řízen:

1. První volanou metodou je **`onPreExecute`**. Tato metoda je provedena na hlavním vlákně a je tedy možné v ní manipulovat s UI. Tato metoda se může použít například na zobrazení UI prvku pro zobrazení postupu.
2. Po `onPreExecute` je volána metoda **`doInBackground`**. Argumentem této metody jsou vstupní parametry a jejím výstupem musí být objekt datového typu, který byl zadán jako result, tedy třetí generický typ AsyncTasku. Tato metoda je prováděna mimo hlavní vlákno a nemůže tedy manipulovat s UI. `DoInBackground` je místem, kde lze podat informace o postupu pomocí metody `publishProgress`, jejímž parametrem je objekt druhého generického datového typu AsyncTasku, progress.
3. **`OnProgressUpdate`** je volána po metodě `publishProgress` a prováděna na hlavním vlákně. Slouží pro aktualizaci postupu.
4. Po dokončení operace a ukončení metody `doInBackground` je volána poslední metoda **`onPostExecute`**. Jejím parametrem je objekt vrácený v `doInBackground`. `OnPostExecute` běží na hlavním vlákně a slouží pro aktualizaci UI získanými daty [50].

```
MyNetworkTask task = new MyNetworkTask();
task.execute();
```

Takto může vypadat kód pro spuštění AsyncTasku. Pokud by třída měla vstupní parametry, byly by zapsány jako argument metody `execute`.

4.8 Uživatelská rozhraní

Všechny prvky uživatelského rozhraní v Android aplikace jsou potomky tříd `View` a `ViewGroup`. `View` zajišťuje zobrazení určitého UI prvku, se kterým může uživatel pracovat, na obrazovku. `ViewGroup` je objekt shromažďující více `View` a `ViewGroup` objektů a definuje jejich rozmístění.

4.8.1 Layouty a prvky UI

Android nabízí několik typů rozmístění UI prvků. Tyto rozmístění se nazývají layouty a mohou být vytvořeny dvěma způsoby. Nejjednodušším a nejpoužívanějším způsobem je deklarace layoutu v XML souboru. Alternativně lze layouty vytvářet programaticky v Javě [51].

Mezi výhody deklarace layoutu v XML patří:

- Rozdělení návrhu UI a aplikační logiky.
- S použitím rozdělení adresářů lze layouty vytvořit pro více verzí systému, nebo více rozměrů, rozlišení a orientací obrazovky bez nutnosti měnit Java kód.
- Lepší vizualizace výsledného UI [51].

Základním typem layoutu je `LinearLayout`. Ve výchozím nastavení skládá všechny obsažené prvky pod sebe, při změně atributu `android:orientation` na hodnotu `horizontal` skládá prvky vedle sebe. Zde je ukázka XML kódu s LinearLayoutem, který obsahuje popisek a tlačítko [51].

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

Element `TextView` představuje popisek, `Button` je tlačítko. Všechny elementy XML musejí mít nastavené atributy `android:layout_height` a `android:layout_width`. Tyto atributy definují výšku a šířku daného elementu a lze je nastavit na tři typy hodnot:

- **Wrap_content** – Přizpůsobí se obsahu elementu. Například šířka `TextView` bude nastavena podle toho, kolik textu obsahuje [52].
- **Match_parent** – Nastaví stejnou velikost jako je velikost rodičovského elementu [52].

- **Konkrétní hodnota**, například 25dp.

Mezi další typy layoutů patří:

- **`RelativeLayout`** – Organizuje elementy relativně k pozicím ostatních elementů. Obsahuje argumenty `android:layout_ below`, `above`, `toLeftOf` a `toRightOf` a další. [53] Lze tak říci například: „Tlačítko 1 je umístěné vpravo od popisku 1 a odsazené vlevo o 6dp“. Za pomoci `RelativeLayout` lze teoreticky docílit libovolného rozmístění prvků, při větším počtu elementů se však stává poměrně nepřehledným.
- **`GridLayout`** – Organizuje prvky do matice. Velikost matice je nastavena argumenty `android:columnCount` a `android:rowCount`. Prvky v tomto layoutu mají nastavené argumenty `android:row` a `android:column`, pomocí kterých je lze přiřadit do libovolného pole v matici [54].
- **`ListView`** je layout představující list položek, kterým lze pohybovat (scrollovat). Položkami `ListView` jsou obvykle layouty s UI prvky, které reprezentují nějaký objekt [55]. Může se jednat například o seznam emailů. Položkou v tomto listu může být například jeden layout obsahující `TextView` pole zobrazující odesílatele, předmět nebo datum odeslání. Data se do `ListView` dostávají pomocí adaptéru, který má na starosti vytvoření layoutu na dané pozici v listu a jeho naplnění odpovídajícími daty. Adaptér poté dynamicky načítá data podle toho, na jaké pozici v listu se uživatel nachází. Pokud by list obsahoval například 1000 položek, adaptér načte pouze položky, které se vejdou na obrazovku a několik dalších pro případ, že uživatel bude s listem pohybovat. Android poskytuje několik typů adaptérů, pro potřeby `ListView` je často používaný `ArrayAdatper` [51].

Získání instance prvku UI v Javě probíhá podobně jako v případě fragmentu. Slouží pro to metoda `findViewById`, kterou lze použít následujícím způsobem:

```
Button tlacitko = (Button) findViewById(R.id.button1);
```

Pomocí uvedeného kódu získáme instanci tlačítka, které má v XML nastavené `android:id` na `button1`.

5 Implementace aplikace Zkoušky

5.1 Napojení na IS/STAG

Informační systém STAG nabízí webové služby, pomocí nichž aplikace získává potřebná data. STAG poskytuje webové služby typu SOAP a REST. Aplikace Zkoušky využívá webové služby REST, protože nabízejí jednoduchý přístup k údajům [56].

5.1.1 Webové služby REST

Webové služby poskytují požadovaná data po zadání URL adresy, která představuje konkrétní službu a metodu. Adresu služby lze rozdělit na čtyři části:

- http://stag-ws.zcu.cz/ws/services/ - Základní adresa webových služeb, která se liší v závislosti na konkrétní škole. Uvedený příklad jsou webové služby ZČU.
- rest/terminy/ - Adresa konkrétní služby. Rest představuje typ služby, terminy je název služby.
- getTerminyProStudenta – Název metody, která získá data.
- ?osCislo=A10B0123P – Parametry vybrané metody [56].

Data lze získat ve formátech XML, XLS, CSV, JSON nebo ICS. V aplikaci je použit formát XML, protože se jedná o populární formát, ze kterého lze jednoduše zpracovat data. Pro nastavení výstupního formátu dat stačí nastavit parametr `outputFormat` na požadovaný formát. Následující adresa vrátí XLS soubor se seznamem termínů studenta:

```
http://stag-ws.zcu.cz/ws/services/rest/terminy/
getTerminyProStudneta?osCislo=A10B0123P&outputFormat=XLS
```

5.1.2 Přihlašování

Pro jednoduché získání dat je tedy možné pouze zadat správnou adresu a přečíst odpověď ve zvoleném formátu. Pro přístup k privátním datům uživatele je však nutné přihlášení. Pro autorizaci uživatele STAG používá standardní metodu HTTP BASIC [56]. Použití této metody spočívá v přidání identifikačních údajů uživatele do hlavičky HTTP požadavku. Identifikačním údajem může být:

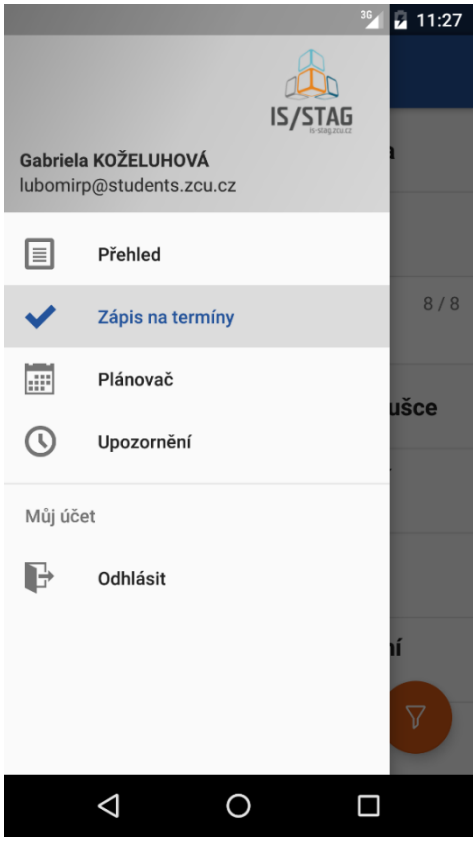
1. Uživatelské jméno a heslo
2. Uživatelův ticket

Ticket uživatele je vygenerovaný unikátní řetězec, který lze získat po úspěšném přihlášení uživatele. Jeho platnost je 60 minut, poté je nutné nové přihlášení a přidělení nového ticketu [56]. Pro účely aplikace Zkoušky je vhodné, aby se uživatel přihlásil pouze jednou a nemusel tak zadávat údaje při každém spuštění, nebo periodicky po vypršení platnosti ticketu. Z toho důvodu je v aplikaci použito HTTP BASIC ověření pomocí zakódované kombinace jména a hesla.

5.2 Uživatelské rozhraní aplikace

Pro navigaci v aplikaci je použit prvek, kterému Android říká Navigation Drawer. Jedná se o menu, které lze vysunout z levé části obrazovky. Tento typ menu je velmi rozšířeným a oblíbeným navigačním prvkem v Android aplikacích, lidově se mu také říká „hamburger menu“.

Obrázek 6: Navigační menu v aplikaci Zkoušky



Zdroj: vlastní zpracování, 2016

Vytváření navigačního menu velmi ulehčuje Android Studio. Protože Navigation Drawer je jedním ze standardních navigačních prvků, Android Studio obsahuje šablonu, pomocí které lze vytvořit aktivitu s tímto menu bez nutnosti psaní jediného řádku kódu.

Za předpokladu, že používáme aktuální verzi Android Studia, šablona obsahuje implementaci odpovídající doporučenému postupu, který lze najít v dokumentaci.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    DrawerLayout drawer = (DrawerLayout)
findViewById(R.id.drawer_layout);
    ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
            this, drawer, toolbar,
R.string.navigation_drawer_open,
R.string.navigation_drawer_close);
    drawer.setDrawerListener(toggle);
    toggle.syncState();

    navigationView = (NavigationView)
findViewById(R.id.nav_view);
    navigationView.setNavigationItemSelectedListener(this);
}
```

Uvedený úryvek kódu je část metody `onCreate` v aktivitě, která zobrazuje navigační menu. Tato aktivita se jmenuje `MainActivity`. Layout aktivity se nachází v souboru activity_main.xml a je nastaven metodou `setContentView`.

```
<android.support.v4.widget.DrawerLayout>
    <include layout="@layout/app_bar_main"></include>
    <android.support.design.widget.NavigationView
        android:id="@+id/nav_view"
        app:headerLayout="@layout/nav_header_main"
        app:menu="@menu/activity_main_drawer" />
</android.support.v4.widget.DrawerLayout>
```

Tento kód je zkrácený layout activity_main.xml, neobsahuje všechny argumenty elementů. `DrawerLayout` je layout, který zaručuje možnost vysunutí menu ze strany obrazovky [57]. Tento layout obsahuje `NavigationView`, což je vlastní navigační menu. `NavigationView` obsahuje dva důležité argumenty.

- **app:headerLayout** – Layout definující vzhled hlavičky menu. Aplikace Zkoušky má v hlavičce zobrazené jméno studenta, jeho emailovou adresu a logo IS/STAG.
- **app:menu** – XML soubor definující položky v navigačním menu.

Vložený layout pod názvem `app_bar_main.xml` obsahuje vlastní layout aktivity.

V `onCreate` se do `DrawerLayoutu` ještě přidává `ActionBarDrawerToggle`. Jedná se o tlačítko v levém horním rohu obrazovky, pomocí kterého lze také vysunout

navigační menu. Metoda `setNavigationItemSelectedListener` nastavuje posluchač kliknutí na položky v navigačním menu.

5.3 Struktura aplikace

Všechny soubory s Java kódem jsou rozděleny do tří balíčků:

1. **eu.polivkal.zkousky** – Balíček obsahuje všechny třídy, které představují Android komponenty. Jedná se hlavně o aktivity, služby a fragmenty.
2. **stag** - Obsahuje třídy, které reprezentují objekty získané ze systému STAG. Jsou zde například třídy `Student`, `Termin` nebo `Predmet`.
3. **data** – Třídy týkající se zpracování a ukládání dat.

5.4 Zpracování dat

Na začátku této kapitoly již byl popsán způsob přístupu k datům v IS/STAG. Tato podkapitola popisuje konkrétní třídy a postupy, které jsou použity pro zpracování dat v aplikaci.

5.4.1 Získání dat

Pro získání dat z IS/STAG musí aplikace komunikovat s webovým serverem na internetu. Pro vývojáře to znamená, že:

- Aplikace musí mít povolení pro přístup k internetu (viz. kapitola 4.1.4 - Oprávnění)
- Operace získání dat nesmí běžet na UI vlákně (viz. kapitola 4.7 - Procesy a vlákna)

Oprávnění pro přístup k internetu je zapsáno v manifestu aplikace.

```
<uses-permission android:name="android.permission.INTERNET" />
```

Protože je potřeba provádět operaci mimo hlavní vlákno, je použit `AsyncTask`. Jelikož aplikace přistupuje k různým datům z IS/STAG, je pro získání a zpracování dat použito několik tříd.

Třída `XmlParser` má za úkol získat žádaná data a vrátit je v požadovaném formátu. Zaslání požadavku na webové služby STAG zajišťují třídy `HttpClient` a `HttpGet`, jejichž použití vypadá následovně:

```
HttpClient httpClient = new DefaultHttpClient();
HttpGet httpGet = new HttpGet(server + "/services/rest/" +
                webServicePath + "/" + method + params);

if (credentials != null) {
    String credentials = this.credentials;
    httpGet.setHeader("Authorization", "Basic " + credentials);
}

HttpResponse httpResponse = httpClient.execute(httpGet);
HttpEntity responseEntity = httpResponse.getEntity();
String s = EntityUtils.toString(responseEntity);
```

5.4.2 Zpracování dat

Zpracování získaných dat je úkolem parseru. Aplikace obsahuje parser pro každý typ objektu, který z IS/STAG získává. Příkladem jsou třídy `StudentParser`, `PredmetParser` nebo `TerminyParser`. Všechny tyto parsery dědí od třídy `XmlParser`. S vlastním zpracováním XML dat pomáhá třída `XmlPullParser`, která dovoluje čtení a pohyb mezi jednotlivými XML tagy a ulehčuje tak vyhledání potřebných dat v XML dokumentu. Parsery vracejí objekty reprezentující zpracovaná data. Může se jednat například o seznam termínů studenta, objektu `Predmet` obsahujícího detailní informace o předmětu nebo seznam osobních čísel studenta.

V předcházejících odstavcích byl popsán způsob získání dat a jejich zpracování do použitelné podoby, stále je však potřeba tyto operace spustit uvnitř AsyncTasku. K tomuto účelu slouží třídy s názvy NetworkTask. V aplikaci se jich opět nachází několik, například `TerminyNetworkTask` nebo `LoginNetworkTask`. Každá z těchto tříd je AsyncTaskem, tj. dědí od třídy `AsyncTask`. Tyto třídy v metodě `doInBackground` volají požadovaný parser a získaná data vrací.

Díky kombinaci uvedených tříd lze zjednodušit a zpřehlednit části kódu, ve kterých dochází ke spuštění operace získání dat. Takto vypadá zápis kódu pro získání termínů přihlášeného studenta.

```
TerminyNetworkTask task = new
TerminyNetworkTask(NetworkTasks.TERMINY_PRO_STUDENTA);
task.setOnPostExecuteListener(this);
task.execute();
```

Parametrem konstruktoru všech NetworkTask tříd je typ operace, který odpovídá konkrétní službě a metodě v IS/STAG. Seznam těchto operací obsahuje `enum` třída

`NetworkTasks`. Podle vybraného typu operace se rozhoduje, která metoda parseru se bude volat.

Účelem použití AsyncTasku je, že po dokončení operace na pozadí se provede aktualizace UI v metodě `onPostExecute`. Protože se však AsyncTasky nenacházejí ve fragmentu, který je zavolal, není v nich možné přistoupit k instancím UI prvků. Tento problém řeší vytvořené rozhraní `OnPostExecuteListener`, které je zakomponované do každé NetworkTask třídy.

```
interface OnPostExecuteListener {
    void onPostExecute(NetworkTasks type, Object o);
}
```

Metoda `onPostExecute` ze zmiňovaného rozhraní má dva argumenty:

- **`NetworkTasks type`** – Typ operace `NetworkTasks`, podle kterého může fragment určit, jaká data byla získána a aktualizovat tak odpovídající prvky UI.
- **`Object o`** – Data vrácená z parserů. Je zde použit obecný typ `Object`, aby bylo možné toto rozhraní použít pro všechny NetworkTasky. Přetypování lze provést na základně typu operace.

Pro použití `OnPostExecuteListener` stačí na objektu NetworkTasku nastavit posluchač metodou `setOnPostExecuteListener` a implementovat rozhraní a jeho metodu `onPostExecute`. Metoda pro nastavení posluchače (`setOnPostExecuteListener`) musí být pochopitelně implementována v každé třídě `NetworkTask`.

5.5 Ukládání dat

Aplikace zkoušky pracuje s následujícími daty, která je potřeba uchovávat:

- Přihlašovací údaje
- Uživatelská data (Jméno, osobní číslo, email..)
- Informace o zkouškových termínech (termíny dále obsahují informace o předmětech)
- Seznam termínů s nastaveným upozorněním na změnu obsazení

Pro ukládání aplikačních dat lze v Androidu využít jednu ze tří možností:

1. Shared Preferences – Slouží k ukládání primitivních dat, běžně pro uživatelská nastavení, ve formátu key-value-pair.
2. SQLite databáze

3. Ukládání na interní paměť

Data ukládaná uvedenými metodami jsou privátní a nelze k nim přistupovat mimo naši aplikaci. Aplikace Zkoušky používá pro ukládání všech dat interní paměť zařízení. Shared Preferences by byly vhodné pro některá data, nelze k nim však přistupovat mimo aplikační komponenty. Použití databáze je zase vhodné zejména pro strukturovaná data, ve kterých je potřeba vyhledávat.

Ukládání a čtení dat zajišťuje třída `AppDataStorageHelper`. Takto vypadá metoda pro čtení aplikačních dat v této třídě.

```
public boolean saveAppData(AppData data) {
    appData = null;
    try {
        FileOutputStream fos =
context.openFileOutput(FILENAME_APPDATA, Context.MODE_PRIVATE);
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(data);
        oos.close();
        fos.close();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
        return false;
    } catch (IOException e) {
        e.printStackTrace();
        return false;
    }
    return true;
}
```

Metoda `openFileOutput` otevře požadovaný soubor v interní paměti zařízení [58]. Práci se soubory zajišťují známé Java třídy jako `FileOutputStream` a `ObjectOutputStream` pro zápis, `FileInputStream` a `ObjectInputStream` pro čtení.

`OpenFileOutput` je volána nad objektem typu `Context`. Jedná se o objekt, který zajišťuje přístup ke třídám a resources dané aplikace [59]. `Context` lze získat z komponent aplikace metodou `getApplicationContext`. Protože `AppDataStorageHelper` není aplikační komponentou, ale samostatnou třídou, potřebuje nějak získat `Context` aplikace.

Nejjednodušším způsobem získání `Contextu` je předání tohoto objektu v konstruktoru. Aplikace Zkoušky k tomu však používá objekt `Application`. `Application` je třída, jejíž instanci vytváří Android při každém spuštění aplikace

[60]. Je možné v ní například uchovávat stavová data, ke kterým je potřeba přistupovat v celé aplikaci, ne jen jedné komponentě. Podobně jako u aktivity je také možné řídit její životní cyklus. Z objektu `Application` lze také získat `Context` metodou `getApplicationContext`. Takto vypadá implementace `Application` v aplikaci.

```
public class MyApp extends Application {
    private static MyApp instance = null;
    public static MyApp getInstance() {
        if (instance == null) {
            instance = new MyApp();
        }
        return instance;
    }
    @Override
    public void onCreate() {
        super.onCreate();
        instance = this;
    }
}
```

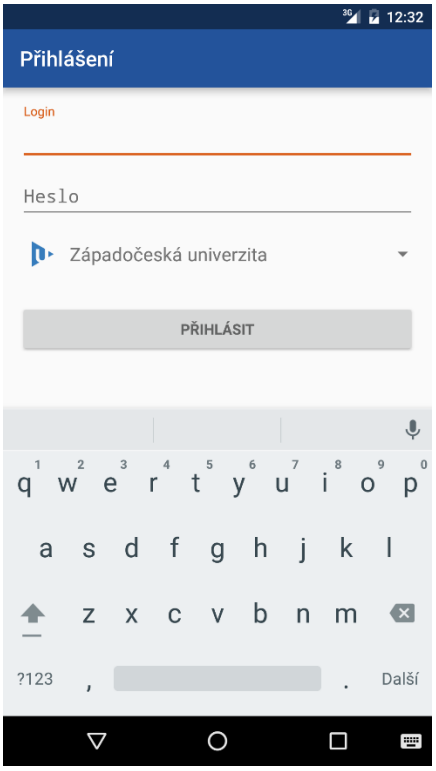
Následné získání aplikačního contextu je provedené v konstruktoru `AppDataStorageHelper` následovně:

```
private AppDataStorageHelper() {
    context = MyApp.getInstance().getApplicationContext();
}
```

5.6 Přihlašování uživatelů

Přihlašování uživatelů je řešeno aktivitou `LoginActivity`. Android Studio obsahuje šablonu aktivity pro přihlášení, ze které je použit základ layoutu.

Obrázek 7: Přihlašovací obrazovka



Zdroj: vlastní zpracování, 2016

Do přihlašovací obrazovky přibylo rozbalovací menu obsahující seznam škol napojených na IS/STAG. Toto menu je řešené UI prvkem `Spinner`. Pro naplnění spinneru se používá `ArrayAdapter`. Po dokončení implementace spinneru a jeho adaptéru by mělo jeho naplnění daty vypadat takto:

```
Spinner spinnerWS = (Spinner)
findViewById(R.id.spinner_wsserver);
WSServerListAdapter spinnerAdapter = new
WSServerListAdapter(getApplicationContext(),
R.layout.row_wsservers_spinner, zakaznikList);
spinnerWS.setAdapter(spinnerAdapter);
```

Prvním krokem je získání instance spinneru již známým způsobem. Dále je třeba inicializovat adaptér a přiřadit jej ke spinneru metodou `setAdapter`. Parametry adaptéru jsou:

- Context
- Layout jednoho prvku spinneru (tj. jak vypadá jeden řádek menu)
- Data

V tomto případě každý řádek v rozbalovacím menu obsahuje název a logo školy. Tyto informace jsou uložené v objektu typu `Zakaznik`. Hlavička adaptéru a jeho konstruktor bude vypadat takto:

```
private class WSServerListAdapter extends ArrayAdapter<Zakaznik>
{
    private Context context;
    private int layoutResource;
    private List<Zakaznik> data = null;

    public WSServerListAdapter(Context context, int resource,
List objects) {
        super(context, resource, objects);
        this.context = context;
        layoutResource = resource;
        data = objects;
    }
```

V adaptéru je nutné překrýt minimálně jednu metodu – `getView`. Tato metoda vrací objekt typu `View`, který představuje jednu řádku rozbalovacího menu naplněnou správnými daty. Argumenty metody `getView` jsou:

- **`int position`** – Pozice prvku v menu, například čtvrtý prvek v menu bude mít nastavenou `position = 3`
- **`View convertView`** – Existující objekt `View` daného prvku. Pokud tento objekt neexistuje (tj. `getView` ještě nebyla volána pro danou pozici), je `convertView` rovno `null`.
- **`ViewGroup parent`** – Rodičovský prvek, do kterého bude vracený `View` objekt vložen.

Takto vypadá metoda `getView` ve zmiňovaném adaptéru.

```
public View getView(int position, View convertView, ViewGroup
parent) {
    if (convertView == null) {
        LayoutInflater inflater = getLayoutInflater();
        convertView = inflater.inflate(layoutResource, parent,
false);
    }
    TextView tvNazev = (TextView)
convertView.findViewById(R.id.row_wsservers_spinner_text_nazev);
    ImageView imageLogo = (ImageView)
convertView.findViewById(R.id.row_wsservers_spinner_image_logo);

    String url = data.get(position).getUrl();
    Zakaznik zakaznik = null;
    for (Zakaznik z : getExistujiciZakaznici()) {
        if (z.getUrl().equals(url)) {
            zakaznik = z;
            break;
        }
    }
    if (zakaznik != null) {
        imageLogo.setImageResource(zakaznik.getLogo());
        tvNazev.setText(zakaznik.getNazevSkoly());
    } else {
        tvNazev.setText(data.get(position).getNazevSkoly());
        imageLogo.setImageDrawable(null);
    }
    return convertView;
}
```

Pokud je `convertView` `null`, je do něj přiřazený odpovídající layout. Používá se k tomu `LayoutInflater`. Jakmile je k dispozici `View` s layoutem, je možné již známým způsobem získat instance UI prvků, které se v něm nacházejí. Do těchto prvků se následně vloží data podle pozice a dojde k vrácení výsledného `View` objektu. Stejný adaptér se používá i pro naplnění `ListView`.

Vlastní přihlášení zajišťuje `LoginNetworkTask`. Tato třída volá webovou službu student a její metodu `getStudentInfo`. Tato metoda vrací informace o studentovi a vyžaduje přihlášení daného studenta. Pokud tato metoda vrátí požadovaná data, je uživatel přihlášen a jeho data jsou uložena do interní paměti. Pokud metoda vrátí prázdný výsledek nebo chybovou hlášku, je přihlášení neúspěšné.

Aplikace ukládá informace o uživateli jako jméno a email a adresu webových služeb vybrané školy. Tyto informace jsou agregované do objektu `AppData`, který je později serializován a uložen do interní paměti zařízení metodou `saveAppData`, jejíž implementace je v podkapitole 5.5 – Ukládání dat.

Přihlašovací údaje uživatele jsou uložené ve vlastním souboru podobným způsobem. Před uložením jsou údaje zakódovány do formátu, ve kterém budou vkládány do HTTP hlavičky. Pro toto zakódování je použita třída `Base64`, jejíž použití vypadá takto:

```
String creds = username + ":" + password;
String creds64 = Base64.encodeToString(creds.getBytes(),
Base64.DEFAULT).replace("\n", "");
```

Ukládané soubory jsou přístupné pouze aplikaci Zkoušky. Ostatní aplikace k nim nemají přístup.

5.7 Zápis na termíny

Zapisování studenta na zkouškové termíny je úkolem fragmentu `TerminyFragment`. Seznam dostupných termínů je zobrazen ve víceúrovňovém rozbalovacím listu. Prvkem, který zajišťuje tuto funkcionalitu, je `ExpandableListView`. Naplnění tohoto prvku daty probíhá podobně jako klasické `ListView` nebo `Spinner`, namísto jedné metody `getView` je však potřeba překrýt `getGroupView` a `getChildView`. Princip je naprosto stejný jako u běžného `ArrayAdapteru`. `GetGroupView` je rodičovský prvek, v tomto případě název předmětu, a `getChildView` představuje podřízený prvek, tedy konkrétní zkouškový termín daného předmětu.

Data o termínech poskytuje metoda `getTerminyProStudenta` ze třídy `TerminyParser`. Pro získání dostupných termínů je nutné, aby byl uživatel přihlášen.

Na zkoušku se lze zapsat kliknutím na vybraný termín. Zobrazí se dialog obsahující detailní informace o termínu. Dialog kromě zápisu a odzápisu umožňuje také nastavit upozornění na změnu obsazení termínu. Tento dialog je ve třídě `ZapisDialogFragment`, která dědí od třídy `DialogFragment`. `DialogFragment` je typ fragmentu umožňující tvorbu dialogů. Tvorba standardního rozhraní dialogu v Androidu se provádí pomocí `AlertDialogu`. Zde je krátký úryvek kódu ze `ZapisDialogFragmentu` zobrazující vytvoření a nastavení dialogu.

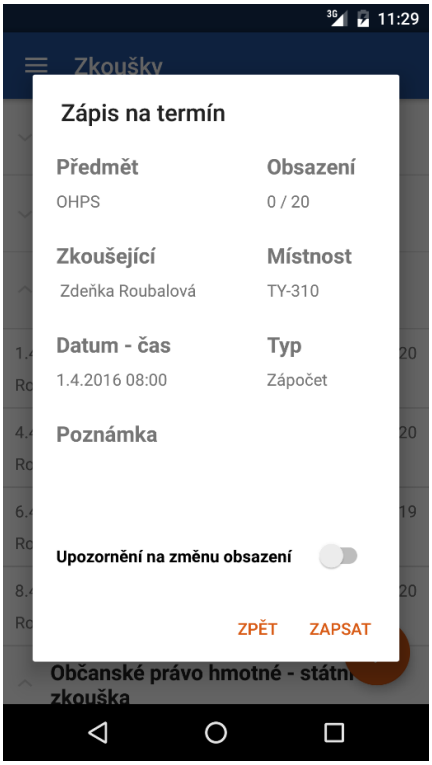
```
public Dialog onCreateDialog(Bundle savedInstanceState) {
    AlertDialog.Builder builder = new
AlertDialog.Builder(getActivity());
    builder.setTitle("Zápis na termín");
    Bundle args = getArguments();
    termin = (Termin) args.getSerializable("termin");

    LayoutInflater inflater = getActivity().getLayoutInflater();
    View view = inflater.inflate(R.layout.dialog_zapis, null);
    builder.setView(view);

    builder.setNegativeButton("Zpět", null);

    return builder.create();
}
```

Obrázek 8: Dialog zápis na termín



Zdroj: vlastní zpracování, 2016

Nejprve je třeba získat instanci `Builderu`, pomocí něhož lze nastavit vlastnosti dialogu. V uvedeném příkladu je nastaven titulek dialogu metodou `setTitle`. Metoda `setView` nastavuje layout, který bude v dialogu použit. Dále lze dialogu nastavit tři typy tlačítek:

- **`setNegativeButton`** – Tlačítko by mělo sloužit na opuštění dialogu bez ukládání změn, např. tlačítko Zpět.
- **`setPositiveButton`** – Tlačítko pro potvrzení, např. tlačítko Zapsat, či OK.

- **`setNeutralButton`** – Používá se, když tlačítko nespadá do výše uvedených kategorií. Příkladem může být možnost odložení akce tlačítkem Připomenout později [61].

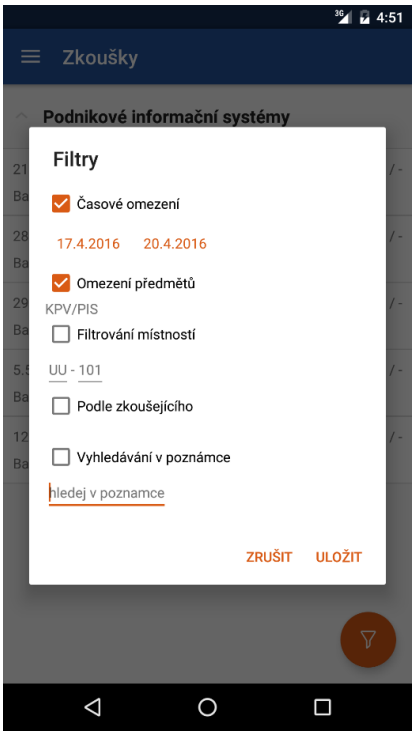
`OnCreateDialog` musí vracet výsledný dialog, jehož instanci lze získat z `Builderu` metodou `create`.

Zobrazené termíny lze také filtrovat. Filtrování se provádí v dialogu, který je vytvořen stejným způsobem jako dialog pro zápis na termín. Protože filtrovací dialog musí vracet data zpět do fragmentu, používá rozhraní `OnFilterDialogDismissListener`.

```
interface OnFilterDialogDismissListener {
    void onFilterDialogDismissed(
            List<Predmet> predmetList,
            List<Ucitel> ucitelList,
            Calendar dateStart,
            Calendar dateStop,
            String budova,
            String mistnost,
            String poznamka
    );
}
```

Vstupní data se do dialogu předávají pomocí objektu `Bundle`, jehož použití je popsáno v kapitole 4.3 – Intent (záměry).

Obrázek 9: Dialog filtry

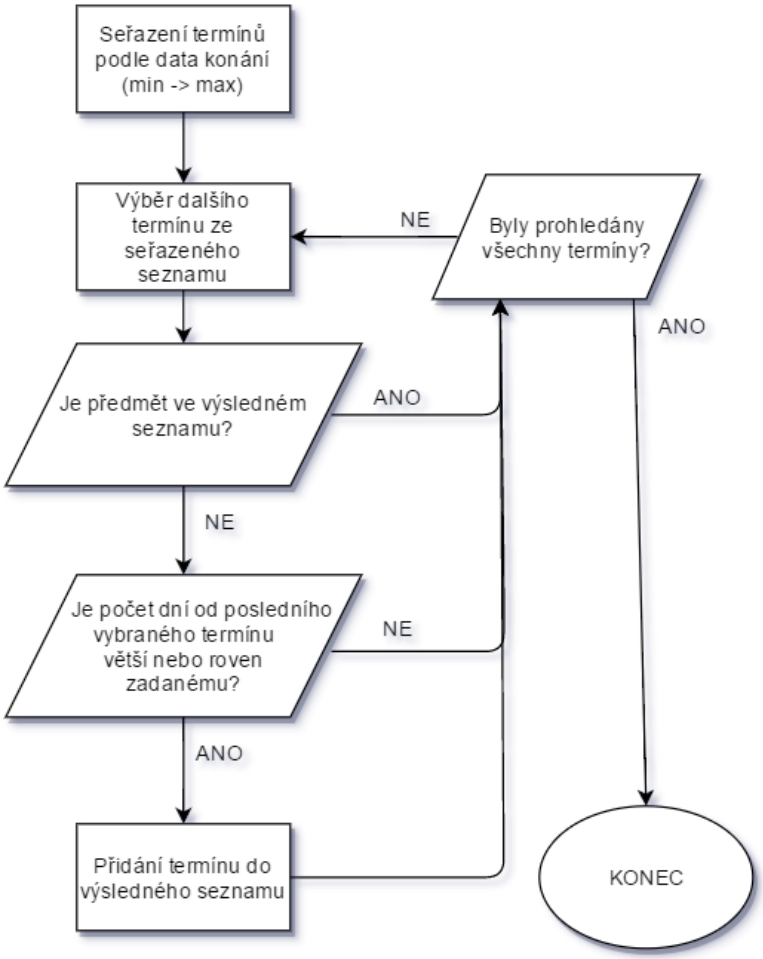


Zdroj: vlastní zpracování, 2016

5.8 Plánovač

`PlanovacFragment` obsahuje velmi jednoduchý layout, který se skládá ze vstupního boxu pro zadání počtu dní mezi zkouškami, tlačítko pro naplánování a `ListView`, ve kterém se zobrazí naplánované termíny. Postup plánování termínů je zobrazen v následujícím diagramu:

Obrázek 10: Postup plánování termínů



Zdroj: vlastní zpracování, 2016

5.9 Upozornění

Záložka upozornění obsahuje seznam všech nastavených upozornění. Upozornění se ukládají do souboru v interní paměti zařízení, opět za pomoci `AppDataStorageHelper`. Konkrétně se ukládají pouze identifikační čísla termínů, na které je nastaveno upozornění.

Pro kontrolování změn v obsazení termínů a rozesílání upozornění je použita klasická spouštěná služba, která každých 10 minut stáhne nová data, porovná je se starými a

případné změny oznámí uživateli prostřednictvím systémové notifikace. Název služby je `NotifikaceService`.

Objekt typu `Handler` zajišťuje spouštění operace každých 10 minut pomocí metody `postDelayed`, která spustí zadaný `Runnable` objekt za zadaný počet milisekund.

```
private Runnable pingTask = new Runnable() {
    public void run() {
        ping();
    }
};
Handler = mHandler = new android.os.Handler();
mHandler.postDelayed(pingTask, 600000);
```

Uvedený kód zobrazuje definici `Runnable` objektu a jeho spuštění s použitím třídy `Handler`. V `NotifikaceService` jsou části tohoto kódu na různých místech.

Princip funkce `NotifikaceService` už byl popsán, jednotlivé operace jsou v kódu rozděleny do několika metod:

- **`onCreate`** – Inicializuje proměnné a spouští metodu ping.
- **`ping`** – Přečte soubor z interní paměti obsahující identifikační čísla termínů, uloží je do proměnné a spustí metodu `getTerminStates`. Pokud je přečtený soubor prázdný, služba se sama ukončí. Nakonec je volána metoda scheduleNext.
- **`getTerminStates`** – Získá aktuální informace o termínech z IS/STAG pomocí `TerminyNetworkTask`. Po dokončení této operace je volána `onPostExecute`.
- **`onPostExecute`** – Porovná získané informace o termínech a vybere pouze takové termíny, na které je nastaveno upozornění. Vybrané termíny uloží do proměnné a zavolá metodu `handleChanges`.
- **`compareTerminStates`** - Porovná aktuální termíny se starými informacemi o termínech. Metoda vrací seznam termínů, u kterých se změnilo obsazení.
- **`handleChanges`** – Nejprve zavolá `compareTerminStates` a získá seznam termínů se změnami. Pro každý termín, ve kterém došlo ke změně, zobrazí systémovou notifikaci s aktuálním obsazením.
- **`scheduleNext`** – Naplánuje další volání metody ping pomocí `postDelayed`.

Samotné zobrazení notifikace zajišťuje metoda `showNotification`.

```
private void showNotification(int id, String title, String
content) {
    NotificationCompat.Builder builder = new
NotificationCompat.Builder(this)
            .setSmallIcon(R.drawable.logo_zcu)
            .setContentTitle(title)
            .setContentText(content)
            .setAutoCancel(true);

    Intent resultIntent = new Intent(this, MainActivity.class);

    TaskStackBuilder stackBuilder =
TaskStackBuilder.create(this);
    stackBuilder.addParentStack(MainActivity.class);
    stackBuilder.addNextIntent(resultIntent);
    PendingIntent resultPendingIntent =
            stackBuilder.getPendingIntent(0,
PendingIntent.FLAG_UPDATE_CURRENT);

    builder.setContentIntent(resultPendingIntent);

    NotificationManager notificationManager =
(NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);

    notificationManager.notify(id, builder.build());
}
```

Jak lze vidět na uvedeném příkladu, pro tvorbu notifikací se používá builder podobný tomu, který se používá pro dialogy. Notifikaci je potřeba nastavit intent, který se provede po kliknutí na danou notifikaci. Jedná se o `PendingIntent`, což je intent, který lze předat jiné aplikaci a umožnit jí spustit komponentu naší aplikace [62]. Notifikace jsou spravovány systémem, přesněji třídou `NotificationManager`. Tato třída tedy potřebuje `PendingIntent` pro spuštění komponenty naší aplikace.

5.10 **Pilotáž**

K provedení pilotáže použití byl použit dotazník System Usability Scale (SUS) [63]. Dotazník má celkem deset jednoduchých otázek, které se hodnotí na stupnici 1 – 5 (rozhodně nesouhlasím – rozhodně souhlasím) a jeho vyplnění zabere zhruba 1 až 2 minuty. SUS byl vybrán z důvodu jednoduchosti a možnosti získání důvěryhodných výsledků i při testování malým množstvím uživatelů [63]. Takto vypadal dotazník, který byl rozeslaný uživatelům aplikace:

1. Rád bych aplikaci používal/a často

2. Aplikace je zbytečně složitá
3. Aplikace se snadno používá
4. Potřeboval/a bych pomoc technické podpory, abych mohl/a aplikaci používat
5. Různé funkce aplikace jsou do ní dobře začleněny
6. Aplikace je příliš nekonzistentní
7. Myslím si, že většina lidí se s aplikací naučí pracovat rychle
8. Aplikace je příliš těžkopádná
9. Při práci s aplikací se cítím velmi jistě
10. Musel/a jsem se naučit hodně věcí, než jsem s aplikací dokázal/a pracovat
11. Uveďte problémy, na které jste narazil/a při používání aplikace (nepovinné)
12. Uveďte své návrhy na vylepšení funkcí aplikace (nepovinné)

Poslední dvě otázky byly přidány za účelem získání konkrétních návrhů a připomínek k aplikaci.

5.10.1 Výsledky

Výsledky SUS lze vyhodnotit následujícím způsobem:

1. Vypočteme skóre pro otázky 1, 3, 5, 7 a 9 tak, že od hodnocení (1 – 5) odečteme jedničku (tj. hodnocení - 1). Tyto otázky tak budou mít hodnocení na stupnici 0 – 4.
2. Vypočteme skóre pro otázky 2, 4, 6, 8 a 10 tak, že hodnocení (1 – 5) odečteme od pětky (tj. 5 - hodnocení). Tyto otázky tak budou mít hodnocení také na stupnici 0 – 4.
3. Všechna vypočtená hodnocení sečteme.
4. Vypočítaný součet vynásobíme koeficientem 2,5 (Maximální skóre je 100).

Dotazník vyplnilo 12 respondentů. Nejnižší hodnocení SUS bylo 85, nejvyšší 97,5 a průměrné hodnocení vyšlo 93,54. Vzhledem k tomu, že maximální možné skóre je 100, se jedná o velmi dobrý výsledek.

Otázku č. 11 nevyplnil žádný respondent. K otázce č. 12 se vyjádřili 3 uživatelé následovně:

- Více možností v plánovači
- Widget, který zobrazuje nejbližší zkoušku
- Plánovač potřebuje více nastavení, takhle není moc užitečný

Závěr

Hlavním cílem této práce byl návrh a implementace mobilní aplikace pro uživatele systému STAG umožňující organizaci zkouškových termínů.

Byla provedena analýza existujících aplikací, které obsahují podobnou funkcionalitu. Následně byl proveden návrh aplikace umožňující zápis a odepsání studenta ze zkouškového termínu a filtrování zobrazených termínů. Součástí návrhu byly také funkce specifické pro tuto aplikaci, kterými jsou možnost upozornění na změnu obsazení termínu a automatické plánování zkouškového období.

Došlo ke srovnání mobilních platforem Android, iOS a Windows Phone. Byly uvedeny možnosti vývoje aplikací na uvedených platformách a popsány výhody a nevýhody těchto systémů z pohledu uživatele i vývojáře. Na základě výsledků srovnání byl pro vývoj aplikace vybrán operační systém Android.

Vlastní implementaci aplikace předcházel popis nejdůležitějších principů používaných pro vývoj Android aplikací. Byla popsána základní adresářová struktura aplikace, její komponenty a jejich použití. Důraz byl kladen na vysvětlení principů, které budou používané při vývoji aplikace.

Poslední kapitola této práce obsahuje popis implementace aplikace Zkoušky. Byly popsány postupy použité pro vývoj důležitých komponent aplikace. Výsledkem byla aplikace splňující požadavky uvedené v návrhu na začátku této práce. Aplikace byla podrobena testování uživateli za účelem ohodnocení použitelnosti. Po vyhodnocení dotazníku bylo zjištěno, že aplikace byla uživateli hodnocena kladně a měla by tak být připravena pro ostré nasazení.

Možnosti rozšíření aplikace v budoucnosti jsou velmi rozsáhlé. Vzhledem k hodnocení uživatelů by bylo vhodné rozšířit funkcionalitu plánovače. Dalším krokem by mohla být optimalizace aplikace pro zařízení s velkým displejem. Aplikace je pro tuto optimalizaci dobře připravena, protože její jednotlivé části jsou implementovány pomocí fragmentů.

Aplikace je publikována v obchodu Google Play Store. Cílem do budoucnosti je aplikaci dále rozvíjet a nabízet tak uživatelům systému STAG pohodlnou organizaci svých zkoušek z mobilního zařízení.

Seznam tabulek a obrázků

Seznam použitých symbolů a zkratek

ADB – Android Debug Bridge

ADT – Android Development Tools

API – Application Programming Interface

AVD – Android Virtual Device

IDE – Integrated Development Environment

IS/STAG – Informační systém studijní agendy

NDK – Native Development Kit

OS – Operační systém

SDK – Software Development Kit

SUS – Systém Usability Scale

UI – User Interface (Uživatelské rozhraní)

XML – Extensible Markup Language

Seznam použité literatury

1. IS/STAG - Další klienti. *IS/STAG - Informační systém studijní agendy.* [Online] [Citace: 30. 3 2016.] https://is-stag.zcu.cz/zajemci/pristupy/dalsi_klienti.html.

2. UniApps. *UniApps.* [Online] [Citace: 30. 3 2016.] http://uniapps.eu/.

3. A History of Windows Phone - MSPoweruser. *MSPoweruser.* [Online] [Citace: 17. 3 2016.] http://mspoweruser.com/a-history-of-windows-phone-the-road-to-threshold/.

4. Number of apps available in leading app stores 2015. *Statista - The Statistics Portal.* [Online] [Citace: 12. 3 2016.] http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/.

5. Create your first app. *Microsoft Dev Center.* [Online] [Citace: 12. 3 2016.] https://msdn.microsoft.com/en-us/windows/uwp/get-started/your-first-app.

6. Get set up. *Microsoft dev center.* [Online] [Citace: 12. 3 2016.] https://msdn.microsoft.com/en-us/windows/uwp/get-started/get-set-up.

7. Account types, locations, and fees. *Microsoft dev center.* [Online] [Citace: 12. 3 2016.] https://msdn.microsoft.com/en-us/windows/uwp/publish/account-types-locations-and-fees.

8. The app certification process. *Microsoft dev center.* [Online] [Citace: 12. 3 2016.] https://msdn.microsoft.com/en-us/windows/uwp/publish/the-app-certification-process.

9. Start developing iOS apps (Swift). *Apple developer.* [Online] [Citace: 12. 3 2016.] https://developer.apple.com/library/ios/referencelibrary/GettingStarted/DevelopiOSAppsSwift/.

10. Choosing a Membership. *Apple Developer.* [Online] [Citace: 12. 3 2016.] https://developer.apple.com/support/compare-memberships/.

11. App Review. *Apple developer.* [Online] [Citace: 12. 3 2016.] https://developer.apple.com/support/app-review/.

12. Historie Androidu v kostce aneb Od verze 1.0 až po Android M. *Svět Androida.* [Online] [Citace: 13. 3 2016.] http://www.svetandroida.cz/historie-androidu-201506.

13. Android NDK. *Android Devepolers.* [Online] [Citace: 13. 3 2016.] http://developer.android.com/tools/sdk/ndk/index.html.

14. Android Developers. *Android Developers.* [Online] [Citace: 13. 3 2016.] http://developer.android.com/.

15. Cross-platform Mobile Web App Development Framework for HTML5 and JS | Sencha. *Design, Develop, and Manage Enterprise Web Applications with Sencha | Sencha.* [Online] [Citace: 17. 3 2016.] https://www.sencha.com/products/touch/#overview.

16. Ionic: Advanced HTML5 Hybrid Mobile App Framework. *Ionic: Advanced HTML5 Hybrid Mobile App Framework.* [Online] [Citace: 17. 3 2016.] http://ionicframework.com/.

17. PhoneGap. *PhoneGap.* [Online] [Citace: 17. 3 2016.] http://phonegap.com/.

18. CSS Components. *Ionic: Advanced HTML5 Hybrid Mobile App Framework.* [Online] [Citace: 17. 3 2016.] http://ionicframework.com/docs/components/#buttons.

19. Material Design. *Introduction - Material design - Google design guidelines.* [Online] [Citace: 17. 3 2016.] https://www.google.com/design/spec/material-design/introduction.html.

20. Xamarin. *Mobile App Development & App Creation Software - Xamarin.* [Online] [Citace: 17. 3 2016.] https://xamarin.com/.

21. IDC: Smartphone OS Marketshare 2015. *IDC.* [Online] [Citace: 12. 3 2016.] http://www.idc.com/prodserv/smartphone-os-market-share.jsp.

22. comScore Reports August 2015 U.S. Smartphone Subscriber Market Share. *comScore.* [Online] [Citace: 12. 3 2016.] https://www.comscore.com/Insights/Market-Rankings/comScore-Reports-August-2015-US-Smartphone-Subscriber-Market-Share.

23. Wha is an Integrated Development Environment. *techopedia.* [Online] [Citace: 17. 3 2016.] https://www.techopedia.com/definition/26860/integrated-development-environment-ide.

24. IntelliJ IDEA the Java IDE. *JetBrains: Development tools for professionals and teams.* [Online] [Citace: 17. 3 2016.] https://www.jetbrains.com/idea/.

25. ADT Plugin Release Notes. *Android Developers.* [Online] [Citace: 22. 3 2016.] http://developer.android.com/tools/sdk/eclipse-adt.html.

26. Android Debug Bridge. *Android Developers.* [Online] [Citace: 23. 3 2016.] http://developer.android.com/tools/help/adb.html.

27. Reading and Writing Logs. *Android Developers.* [Online] [Citace: 23. 3 2016.] http://developer.android.com/tools/debugging/debugging-log.html.

28. Using the Emulator. *Android Developers.* [Online] [Citace: 23. 3 2016.] http://developer.android.com/tools/devices/emulator.html.

29. Managing Virtual Devices. *Android Developers.* [Online] [Citace: 23. 3 2016.] http://developer.android.com/tools/devices/index.html.

30. Features - Genymotion Android Emulator. *Genymotion Android Emulator.* [Online] [Citace: 30. 3 2016.] https://www.genymotion.com/features/.

31. Pricing & Licencing - Genymotion Android Emulator. *Genymotion Android Emulator.* [Online] [Citace: 30. 3 2016.] https://www.genymotion.com/pricing-and-licensing/.

32. Managing Projects Overview. *Android Developers.* [Online] [Citace: 3. 4 2016.] http://developer.android.com/tools/projects/index.html.

33. Providing Resources. *Android Developers.* [Online] [Citace: 3. 4 2016.] http://developer.android.com/guide/topics/resources/providing-resources.html.

34. Supporting Multiple Screens. *Android Developers.* [Online] [Citace: 3. 4 2016.] http://developer.android.com/guide/practices/screens_support.html.

35. Building a Simple User Interface. *Android Developers.* [Online] [Citace: 3. 4 2016.] http://developer.android.com/training/basics/firstapp/building-ui.html.

36. Accessing Resources. *Android Developers.* [Online] [Citace: 3. 4 2016.] http://developer.android.com/guide/topics/resources/accessing-resources.html.

37. System Permission. *Android Developers.* [Online] [Citace: 3. 4 2016.] http://developer.android.com/guide/topics/security/permissions.html.

38. Requesting Permissions at Run Time. *Android Devleopers.* [Online] [Citace: 3. 4 2016.] http://developer.android.com/training/permissions/requesting.html.

39. App Manifest. *Android Developers.* [Online] [Citace: 3. 4 2016.] http://developer.android.com/guide/topics/manifest/manifest-intro.html.

40. Application Fundamentals. *Android Developers.* [Online] [Citace: 4. 4 2016.] http://developer.android.com/guide/components/fundamentals.html.

41. Intents and Intent Filters. *Android Developers.* [Online] [Citace: 4. 4 2016.] http://developer.android.com/guide/components/intents-filters.html.

42. Intent. *Android Developers.* [Online] [Citace: 4. 4 2016.] http://developer.android.com/reference/android/content/Intent.html.

43. Taking Photos Simply. *Android Developers.* [Online] [Citace: 4. 4 2016.] http://developer.android.com/training/camera/photobasics.html.

44. Starting Another Activity. *Android Developers.* [Online] [Citace: 5. 4 2016.] http://developer.android.com/training/basics/firstapp/starting-activity.html.

45. Activities. *Android Developers.* [Online] [Citace: 5. 4 2016.] http://developer.android.com/guide/components/activities.html.

46. Fragments. *Android Developers.* [Online] [Citace: 10. 4 2016.] http://developer.android.com/guide/components/fragments.html.

47. Services. *Android Developers.* [Online] [Citace: 10. 4 2016.] http://developer.android.com/guide/components/services.html.

48. Service. *Android Developers.* [Online] [Citace: 10. 4 2016.] http://developer.android.com/reference/android/app/Service.html#onUnbind(android.content.Intent).

49. Processes and Threads. *Android Developers.* [Online] [Citace: 10. 4 2016.] http://developer.android.com/guide/components/processes-and-threads.html.

50. AsyncTask. *Android Developers.* [Online] [Citace: 10. 4 2016.] http://developer.android.com/reference/android/os/AsyncTask.html.

51. Layouts. *Android Developers.* [Online] [Citace: 11. 4 2016.] http://developer.android.com/guide/topics/ui/declaring-layout.html.

52. Layout Resource. *Android Developers.* [Online] [Citace: 11. 4 2016.] http://developer.android.com/guide/topics/resources/layout-resource.html#layoutvalues.

53. Relative Layout. *Android Developers.* [Online] [Citace: 11. 4 2016.] http://developer.android.com/guide/topics/ui/layout/relative.html.

54. GridLayout. *Android Developers.* [Online] [Citace: 11. 4 2016.] http://developer.android.com/reference/android/widget/GridLayout.html.

55. List View. *Android Developers.* [Online] [Citace: 11. 4 2016.] http://developer.android.com/guide/topics/ui/layout/listview.html.

56. Webové služby IS/STAG. *Webové služby IS/STAG.* [Online] [Citace: 11. 4 2016.] https://stag-ws.zcu.cz/ws/web?pp_locale=cs&pp_reqType=render&pp_page=tech.

57. DrawerLayout. *Android Developers.* [Online] [Citace: 13. 4 2016.] http://developer.android.com/reference/android/support/v4/widget/DrawerLayout.html.

58. Storage Options. *Android Developers.* [Online] [Citace: 16. 4 2016.] http://developer.android.com/guide/topics/data/data-storage.html#db.

59. Context. *Android Developers.* [Online] [Citace: 16. 4 2016.] http://developer.android.com/reference/android/content/Context.html.

60. Application. *Android Developers.* [Online] [Citace: 15. 4 2016.] http://developer.android.com/reference/android/app/Application.html.

61. Dialogs. *Android Developers.* [Online] [Citace: 16. 4 2016.] http://developer.android.com/guide/topics/ui/dialogs.html.

62. PendingIntent. *Android Developers.* [Online] [Citace: 16. 4 2016.] http://developer.android.com/reference/android/app/PendingIntent.html.

63. System Usability Scale (SUS). *usability.gov.* [Online] [Citace: 16. 4 2016.] http://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html.

Seznam příloh

Příloha A: Obsah přiloženého CD

Příloha A

Obsah přiloženého CD:

- zdrojovy_kod – Složka obsahuje zdrojový kód vytvořené aplikace.
- aplikace – Obsahuje vyexportovaný soubor s příponou .apk pro instalaci aplikace.
- dotaznik – Obsahuje soubor s kompletními výsledky testování.

Abstrakt

POLÍVKA, Lukáš. *Vývoj aplikací pro mobilní telefony.* Plzeň, 2016. 70 s. Bakalářská práce. Západočeská univerzita v Plzni. Fakulta ekonomická.

Klíčová slova: Android, vývoj aplikací, mobilní aplikace, zkoušky, IS/STAG

Předložená bakalářská práce je zaměřena na návrh a implementaci mobilní aplikace pro platformu Android. Jedná se o aplikaci, která uživatelům systému IS/STAG umožňuje organizaci zkouškových termínů. V první části této práce je proveden návrh aplikace, srovnání nejrozšířenějších operačních systémů pro mobilní telefony a je vybrána vhodná platforma pro vývoj aplikace. Práce se dále zabývá popisem základních principů vývoje aplikací pro Android. V poslední části je popsána vlastní implementace aplikace a je provedeno testování použitelnosti. Výstupem práce je mobilní aplikace pro systém Android, která by měla sloužit studentům pro pohodlné zapisování na zkoušky ze svého mobilního telefonu.

Abstract

POLÍVKA, Lukáš. *Application development for mobile phones.* Plzeň, 2016. 70 s. Bachelor Thesis. University of West Bohemia. Faculty of Economics.

Key words: Android, application development, mobile applications, exams, IS/STAG

This bachelor thesis focuses on design and implementation of a mobile application for the Android platform. It is an application that enables users of IS/STAG to organize their examinations. The first part of this thesis contains the design of the application, compares popular mobile operating systems and selects a suitable platform for development of this application. This thesis also describes the basic principles of application development for Android. The last part describes the actual implementation of the application and conducts usability testing. The output of this thesis is a mobile application for Android, which should serve students for comfortable registration for examinations from their mobile phone.