VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MOBILNÍ APLIKACE PRO PROHLÍŽENÍ INTERNETOVÝCH OBRÁZKŮ

MOBILE APP FOR VIEWING INTERNET PICTURES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **DAVID KOČNAR**
AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. ADAM HEROUT, PhD.**
SUPERVISOR

BRNO 2018

Zadání bakalářské práce/20496/2017/xkocna01

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Kočnar David**
Obor: Informační technologie
Téma: **Mobilní aplikace pro prohlížení internetových obrázků**
Mobile App for Viewing Internet Pictures
Kategorie: Uživatelská rozhraní

Pokyny:

1. Seznamte se s problematikou vývoje aplikací pro mobilní zařízení; zaměřte se na platformu Android.
2. Vyhledejte a analyzujte existující aplikace pro prohlížení obrázků a pro konzumaci vizuálního obsahu.
3. Prototypujte dílčí prvky uživatelského rozhraní řešené aplikace a testujte je na uživatelích.
4. Navrhněte a implementujte aplikaci pro prohlížení online obrázků na základě zájmů uživatele.
5. Navrhněte a implementujte algoritmy výběru, vyhledávání a doporučování obrázků.
6. Testujte řešenou aplikaci na uživatelích a iterativně ji vylepšujte.
7. Zhodnoťte dosažené výsledky a navrněte možnosti pokračování; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Steve Krug: Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability, ISBN-13: 978-0321965516
- Android Developers: https://developer.android.com/index.html
- Susan M. Weinschenk: 100 věcí, které by měl každý designér vědět o lidech, Computer Press, Brno 2012

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3, značné rozpracování bodů 4 a 5.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Herout Adam, prof. Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato práce vysvětluje problematiku vývoje mobilních aplikací pro operační systém Android a popisuje kompletní návrh a vývoj mobilní aplikace Impres pro prohlížení obrázků. Kromě mobilní aplikace byla vytvořena také serverová aplikace v jazyce PHP zajišťující data z více internetových zdrojů. Celá tato služba má za cíl přinést uživateli inspirativní obsah tvořený fotografiemi a dalším vizuálním uměním. Služba bude automaticky doplňovat nový obsah a pro uživatele vytvářet personalizovaný výběr. V práci jsou popsány teoretické znalosti a postupy nutné pro správný návrh a vývoj mobilních aplikací. Závěrem práce je zhodnocení řešení a návrhy na případné další funkce.

Abstract

This thesis explains issues concerning the development of mobile applications for Android and describes a complete design and development of mobile application Impres for image viewing. In addition to the mobile application there was also created a server application in PHP providing data from different internet sources. Taken together this service aims to bring to its user an inspiring content consisting of phototography and other visual art. The service will automatically add new content and create user-personalised selection. The work also describes theoretical knowledge and procedures necessary for technical design and development of mobile applications. Conclusion evaluates solution and proposals for possible other features.

Klíčová slova

mobilní aplikace, Android, obrázky, umění, fotografie, inspirace, API, server

Keywords

mobile application, Android, images, art, photograph, inspiration, API, server

Citace

KOČNAR, David. *Mobilní aplikace pro prohlížení internetových obrázků*. Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, PhD.

Mobilní aplikace pro prohlížení internetových obrázků

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana prof. Adama Herouta. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
David Kočnar
16. května 2018

Poděkování

Chtěl bych poděkovat vedoucímu práce – prof. Heroutovi za výborné postřehy a pravidelné konzultace, které byly vždy přínosem.

Obsah

Kapitola 1

Úvod

Současně s růstem popularity mobilních zařízení se měnil i způsob, jakým uživatelé tato zařízení používají. Od základních funkcí jako telefonování a odesílání textových zpráv se účel posunul spíše k všestrannému kapesnímu asistentovi. Důležitou roli ale hraje také zábava a možnost si kdykoli a kdekoli zlepšit náladu či zpříjemnit volnou chvíli.

Tato práce popisuje návrh a vývoj mobilní aplikace Impres pro operační systém Android. Aplikace slouží k zobrazování obrázků ve vysokém rozlišení z několika internetových zdrojů v jednotném uživatelském prostředí. Zaměřuje se na jednoduchý moderní vzhled a využití prvků personalizace obsahu. Důvodem pro vypracování bylo vytvořit kapesní řešení poskytující inspiraci a zábavu formou kvalitních obrázků. Po dalším studiu problematiky psychologie umění jsem si uvědomil, že se na dané téma dá pohlížet více způsoby. Jedná se o snahu zprostředkovat uživateli prožitky podobné těm, které zažívá v galeriích, muzeích nebo například při cestování. Tedy věcí, které obvykle pomáhají k udržování inspirace a kreativity. Rozhodl jsem se tedy zaměřit na přinesení inspirace a emočního prožitku kontaktem s vizuálním uměním a kvalitními fotografiemi z celého světa, jak blíže popisuji v kapitole 2.

Cílem aplikace je poskytovat inspirativní vizuální obsah v uživatelsky přívětivé aplikaci. Určil jsem si proto několik vlastností, které by výsledná aplikace měla splňovat. Je to zejména jednoduché použití, animace a gesta pro dotvoření celkového pocitu z aplikace. Potom také systém doporučování obsahu a současně zachování anonymity. Nejedná se o službu s prvky sociální sítě. Kromě toho chci umožnit používání aplikace i bez přístupu k internetovému připojení, což bude možné v omezeném režimu.

Aplikace Impres je určena zejména pro milovníky fotografií, lidi působící v kreativních odvětvích a fotografy. Je vhodná ale i jako volnočasová aplikace pro zlepšení nálady či získání inspirace. Součástí této práce je i průzkum trhu a krátké pojednání o vymezení aplikace Impres na současném trhu mobilních aplikací.

Práce je členěna do 9 kapitol. Záměrem bylo popsat motivaci pro tuto práci. Následně shrnout teoretické informace týkající se vývoje mobilních aplikací pro operační systém Android. Kromě základních informací jsem se zaměřil i na architektonické návrhy a v samostatné kapitole popsal několik použitých knihoven. Z těchto témat vycházím při návrhu a vývoji samotné aplikace Impres. Kromě té bude popsán i vývoj serverové aplikace v jazyce PHP. Tento server má na starost získávání dat z více služeb, ukládání informací a poskytování sjednocených dat klientské aplikaci. Z toho důvodu je popsána i problematika práce s webovým API.

Kapitola 2

Motivace k vytvoření aplikace

Vycházím z názoru, že podobně jako kvalitní hudba dokáže ovlivňovat lidské emoce a znatelně se liší emoční prožitek ve srovnání s hudbou nekvalitní, tak i vizuální umění může ovlivňovat lidskou psychiku a emoce. Jaké jsou motivy k recepci[1] umění blíže popisuje Jiří Kulka [10, strana 388]. Zmiňuje zejména aktivizaci emocionální sféry — touhu po citových a také estetických zážitcích, případně i hledání smyslu života či aktivizaci intelektuální sféry. Ovšem řadí mezi ně i prostší motivy jako oddech či zábavu.

Zásadním termínem této práce je inspirace, tedy podnět, nápad, vnuknutí či nával kreativity. Oxfordský slovník používá doslovný výklad „nadechnout“ a přeneseně: „Prodchnutí nebo proniknutí nějaké myšlenky, záměru atd. do mysli.“ Wurmanová [16] také uvádí, že inspirace k nám přichází nečekaně a bez vědomého záměru. Ovšem zmiňuje, že lidé přicházející více do styku s inspirujícími podněty, například uměleckými díly, mohou zažívat inspiraci častěji. Dále jsou to údajně lidé otevření novým zážitkům a ti věnující se tvůrčí činnosti.

Právě jako tradiční zdroje inspirace tedy můžeme uvést galerie, muzea, případně přírodu a cestování. Tyto zdroje jsou pravděpodobně nenahraditelné spolu s mnohými dalšími. Motivací pro aplikaci Impres je právě, alespoň částečné, přenesení těchto prožitků do chytrého mobilního telefonu. Zprostředkovává kontakt s uměleckými díly a také kvalitními uměleckými fotografiemi. Uživatel tedy může poznávat celý svět tzv. skrze objektiv, nebo skrze umělcovo plátno.

2.1 Zdroje obrázků

Vizuální umění a fotografie může ovlivňovat a ovlivňuje lidské emoce a záleží na jeho kvalitě. Je proto zásadní zaměřit se právě na zdroje kvalitních fotografií a dalších obrázků. Primárním zdrojem uměleckých i dokumentárních fotografií jsou profesionální fotografové, ale můžou mezi ně patřit i amatéři. Tito fotografové často využívají specializované servery a sociální sítě, kde sdílí své fotografie. Mezi nejznámější patří 500px.com, Unsplash.com nebo Flickr.com.

Vizuální umění je třeba rozdělit na klasické výtvarné umění a digitální tvorbu. První skupina se obvykle nachází fyzicky v galeriích a na výstavách, nicméně některé galerie nabízí přístup ke své sbírce i online. Druhá skupina bývá již tvořena za účelem prezentování online. Proto, podobně jako u fotografů, i tito umělci často prezentují své výtvory na speci-

[1] Recepce je schopnost přijímat podněty smyslovými orgány.

alizovaných serverech. Řada z těchto služeb poskytuje veřejné REST API[2], proto je z nich možné data získávat automatizovaným způsobem.

[2]REST API (Representational state transfer – Application interface) je architektura pro provádění požadavků a získávání odpovědí skrze protokol HTTP. Základy této problematiky popisuje Deering [7].

Kapitola 3

Existující aplikace a služby

V současné době existuje velké množství aplikací a služeb, proto je analýza trhu velmi důležitá. Jen Google Play obsahuje přes 2,8 milionů aplikací ke stažení [1]. Často ovšem aplikace nedosahují vysokých kvalit anebo neplní dobře ani základní potřeby uživatele. Proto právě důraz na kvalitu a ergonomii použití může značně ovlivnit popularitu a komerční úspěch aplikace.

Protože je moje aplikace určena pro operační systém Android, zaměřím se na analýzu aplikací pro tento systém. Aplikace s podobným zaměřením by se daly rozdělit do dvou skupin. Tou první budou přímo služby poskytující obsah, které mají často i vlastní mobilní aplikaci. Druhá skupina pouze zobrazuje obrázky z jiného zdroje. Zejména v první skupině je možné očekávat kvalitnější a modernější aplikace, proto se na ně zaměřím konkrétněji.

3.1 Flickr

Flickr.com je komunitní webová služba sdružující amatérské, ale i profesionální fotografy. Umožňuje nahrávat ve vysoké kvalitě pořízené fotografie, organizovat, upravovat a sdílet. Fotografie je možné nastavit jako veřejné pod licencí umožňující další použití, nebo jako soukromé.

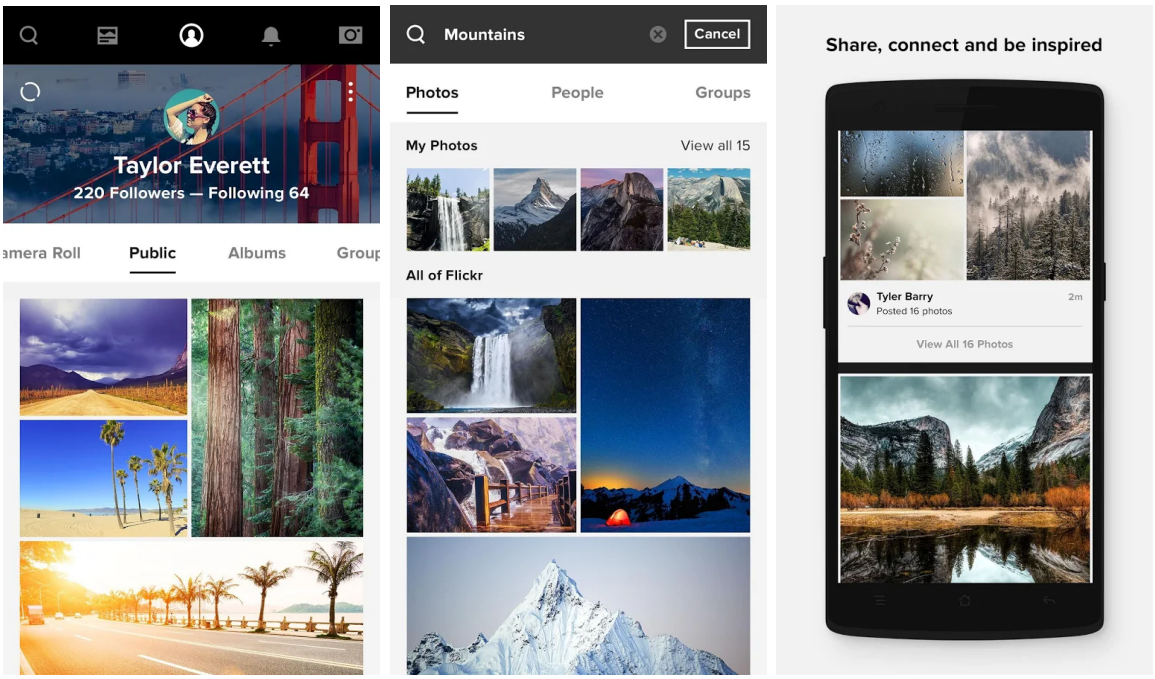
Flickr také poskytuje veřejné REST API[1]. To obsahuje veliké množství různých metod pro přihlášení, kontakty, kolekce, blogy, galerie, skupiny, místa, ale samozřejmě také pro získání uživatelů a fotografií. Speciální metoda pro získávání vybraného zajímavého obsahu pro každý den je nazvána “Interestingness” (česky zajímavost). Odpověď s daty je ve formátu XML, ale lze snadno změnit přepínačem na JSON. Podobně jsou dostupné další přepínače pro rozšiřující funkce API. Pro provedení každého dotazu je vyžadován klíč označovaný “api_key”. Pouze pokud by bylo potřeba přes API provádět operace dostupné přihlášenému uživateli, je vyžadována nejdříve autentizace protokolem OAuth.

Tato služba má vlastní mobilní aplikaci pro operační systém Android[2]. Pro její používání je nutno se nejdříve registrovat a přihlásit. Aplikace zpřístupňuje veškerou funkcionalitu služby, nahrávání, organizování a také úpravy fotografií. Umožňuje automatické nahrávání fotografií a videí. Prohlížení obsahu celé komunity je samozřejmou součástí. Flickr uvádí, že jsou takto přístupné miliony skupin a miliardy fotografií. Aplikace využívá moderní prvky systému Android i některé principy designového jazyku Material Design. Neobvykle zvolená je hlavní navigace v aplikaci. Na horním okraji se nachází lišta s ikonami pro přepínání

[1] https://www.flickr.com/services/api/

[2] https://play.google.com/store/apps/details?id=com.yahoo.mobile.client.android.flickr

hlavních obrazovek. Material Design ovšem takovouto lištu doporučuje u spodního okraje a je pro to obvykle v aplikacích využíván prvek `BottomNavigationBar`. Zde se jedná o méně obvyklou a na velkých displejích potenciálně méně ergonomickou variantu.



Obrázek 3.1: Ukázka mobilní aplikace služby Flickr.com. Převzato z: https://play.google.com/store/apps/details?id=com.yahoo.mobile.client.android.flickr

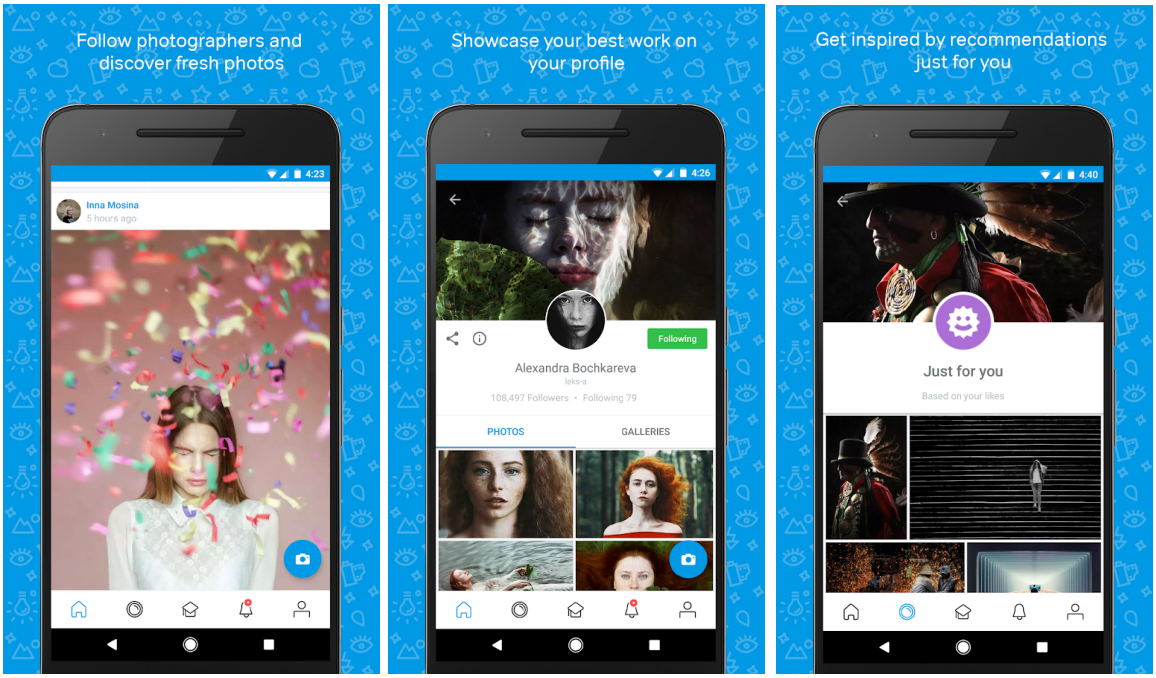
3.2 500px

Webová služba 500px.com se zaměřuje na prezentování práce fotografů. Kromě nahrávání fotografií a prohlížení práce ostatních, umožňuje také svým uživatelů vydělávat peníze. Pro plnou funkčnost služby lze vybrat jeden z placených programů.

Je možné využít také veřejné REST API, to ovšem obsahuje nějaká omezení. Obrázky poskytované skrze toto API mají nižší rozlišení, maximálně 900 pixelů v obou osách. Což pro mnoho způsobů použití nemusí být vhodné.

Služba 500px využívá vlastní mobilní aplikaci pro systém Android[3]. Ta zobrazuje obrázky v použitelném rozlišení a obsahuje řadu funkcí. Nicméně není ji možné používat bez registrace. Umožňuje zobrazení nejnovějších fotografií sledovaných fotografů. V obchodě Google Play je ukázka obrazovky s vybranými obrázky pole dříve označených fotografií tlačítkem "Líbí se mi" (symbol srdce), aktuálně ale tato obrazovka v aplikaci není. Aplikaci nelze nijak používat bez připojení k internetu, protože neobsahuje funkcionalitu přednačítání dat.

[3] https://play.google.com/store/apps/details?id=com.fivehundredpx.viewer



Obrázek 3.2: Mobilní aplikace 500px. Základní zobrazení aktuálních fotografií a detail profilu fotografa. Převzato z: https://play.google.com/store/apps/details?id=com.fivehundredpx.viewer

3.3 Unsplash

Služba Unsplash.com slouží ke sdílení profesionálních fotografií bez autorských práv. Podle statistik[4] služba obsahuje přes 500 000 fotografií od více než 80 000 fotografů.

REST API této služby umožňuje řadu využití. Lze získávat seřazené fotografie podle parametrů, nebo například využit koncový bod pro kurátorované (curated) fotografie. API vrací adresy na obrázky ve vysoké kvalitě.

Mobilní aplikaci pro prohlížení obrázků přímo služba neprovozuje. Jediná oficiální aplikace se jmenuje Unsplash Wallpapers a slouží k automatickému vyměňování tapety na pozadí. Nicméně existují neoficiální aplikace, které se více či méně snaží napodobit funkcionalitu Unsplash.com. Podle hodnocení i počtu instalací vede aplikace Resplash, ale obsahuje jen základní funkce.

3.4 Artsy

Databáze uměleckých děl Artsy.net je zaměřená zejména na sběratele umění. Tato služba poskytuje umělecká díla, sbírky a další typy obsahu přes své API. Obrazy jsou dostupné v použitelném rozlišení. Žádná oficiální mobilní aplikace Artsy pro systém Android neexistuje. Nenašel jsem ani žádnou neoficiální, u které by bylo možno dohledat, že pracuje s touto databází.

[4] https://unsplash.com/stats

3.5 Pexels

Služba Pexels.com poskytuje mnoho kvalitních fotografií ke stažení zdarma. Většina fotografií je dostupná ve velmi vysokém rozlišení. Pexels provozuje také REST API, přes které je možno tyto fotografie procházet. Poskytovaná data ale neobsahují klíčová slova, proto není možné snadno automatizovaně určit obsah obrázků. Služba nemá žádnou oficiální aplikaci pro systém Android. Je nicméně pravděpodobné, že její data využívají některé jiné aplikace.

3.6 DeviantART

DeviantArt je služba zaměřená na amatérské tvůrce digitálních uměleckých děl. Komunita čítá více než 26 milionů uživatelů. Značnou část obsahu tvoří digitální ilustrace žánru fantasy. K obsahu je možno automatizovaně přistupovat přes REST API nebo veřejné RSS kanály. Tato služba provozuje i vlastní mobilní aplikaci, která se zaměřuje na prohlížení prací i udržování kontaktu členů komunity.

3.7 Další aplikace s obrázky

Aplikací a služeb nějak pracujících s obrázky a fotografiemi existuje samozřejmě mnoho dalších. Každá se ovšem zaměřuje na trochu něco jiného. Veliká skupina aplikací se zaměřuje na tapety plochy systému. V této skupině například aplikace Wallpaper HD, Backgrounds nebo Walli. Zejména pak poslední jmenovaná vybočuje z řady, protože poskytuje vlastní kreativní obsah. Jestli některá aplikace využívá více zdrojů obsahu, nelze jistě určit.

3.8 Zhodnocení prostoru na trhu

Jak je výše zmíněno, aplikací pracujících s internetovými obrázky existuje mnoho. Na základě analýzy současných řešení jsem dospěl k několika závěrům. Z výše zmíněných služeb je možné využívat data pro aplikaci Impres z API Flickr.com, Unsplash.com, Artsy.com a DeviantART.com. Ostatní buďto neposkytují přes API dostatečnou kvalitu obrazu, nebo neobsahují vhodný formát dat, případně by toto využití API bylo v rozporu s podmínkami. V budoucnu je možno provádět analýzu dalších zdrojů dat, např. méně známých služeb, a přidávat tak další data pro aplikaci Impres.

Některé z aplikací obsahují zajímavou funkcionalitu a propracované ovládání. Je zřejmé, že za nimi stojí profesionální tým vývojářů. Proto je možné se inspirovat dobře fungujícími prvky výše zmíněných aplikací. Dobře použitelnou se jeví hlavní navigace ve formě lišty, kde je možné přepínat mezi hlavními částmi aplikace. Tato lišta by ale měla nejen dobře vypadat, ale být také vhodně umístěna. Vhodná je možnost označovat fotografie tlačítkem signalizujícím, že se nám obrázek líbí. Například na základě takto označených obrázků je následně možné zobrazovat doporučený obsah.

Naopak tyto aplikace mají i méně vhodné vlastnosti či chybějící funkce. U aplikací 500px a Flickr je nutné se nejdříve registrovat a přihlásit svými údaji v mobilní aplikaci. Jinak uživatel aplikaci nemůže používat. Dále, jak aplikace 500px, tak Flickr nedokáže nijak pracovat bez připojeného internetu. Neprovádí žádné přednačtení obsahu, který by tak byl dostupný v omezeném režimu i bez internetu. Když budeme hodnotit vizuální stránku aplikací, není toho moc k vytčení, nicméně chybí zde plynulé animace při prohlížení obrázků. Uživatel mezi jednotlivými obrazovkami neprochází plynule, ale dojde k bliknutí celé ob-

razovky. Dále není možné využít některá z běžněji používaných uživatelských gest. Těmito gesty by bylo možné aplikaci ovládat například i jedním prstem, a tak zvýšit komfort a ergonomii aplikace. Například potažení prstem nahoru či dolů pro návrat a zavření obrazovky s detailem.

Ty aplikace, které se snaží poskytovat obsah z jiných zdrojů, se většinou zaměřují spíše na jednoduché tapety, nebo nemají vysokou obrazovou kvalitu či neaktualizují databázi obsahu. Naopak aplikace vytvořené přímo poskytovateli obsahu jsou obvykle dobrou ukázkou funkčnosti a možnou inspirací. Moje aplikace Impres tak může využít výhody i nevýhody z obou skupin a doplněním přidané hodnoty ještě zvýšit šanci na úspěch. To, čím se záměr aplikace zejména odlišuje, je zaměření na kombinaci více forem obrazového obsahu. Výše zmíněné aplikace se téměř na propojení kvalitních fotografií a současné i dalších forem vizuálního umění nezaměřují.

Kapitola 4

Vývoj aplikací pro operační systém Android

Vývoj mobilních aplikací lze provádět buďto nativně nebo pomocí technologií umožňujících multiplatformní vývoj. Nejrozšířenější operační systémy jsou Android a Apple iOS. Pro první zmíněnou platformu se využívá jazyk Java či Kotlin s Android SDK, je ale možné využít i C/C++ díky Android NDK. Pro platformu iOS se využívají jazyky Objective-C nebo Swift. Existují ovšem technologie umožňující vývoj aplikací pro více platforem současně, jako Xamarin (C#), PhoneGap (JS), React Native (JS) a další. Ty jsou vhodné zejména pro jednoduché aplikace. [15]

Blíže budu popisovat operační systém Android a vývoj v jazyce Java, protože jsem tuto technologii zvolil pro vypracování aplikace Impres.

Android je open-source mobilní operační systém s jádrem Linux. Postupně se rozšířil i na tablety, chytré hodinky či televize. Proto je u něj důležitá všestrannost a univerzálnost, na což je vhodné pohlížet i při vývoji aplikací. V jazyce Java se aplikace vytváří s použitím `Java API framework`. Tento kód je zkompilován do byte kódu, aby mohl být spuštěn ve virtuálním stroji. Aktuální verze systému Android využívají běhové prostředí `Android Runtime` (ART), před verzí 5.0 to byl `Dalvik Virtual Machine`. Další detaily o architektuře této platformy je možné najít v příručce [4].

4.1 Základní komponenty a třídy

Základní bloky tvořící aplikaci, jsou `Activity`, `Service`, `Content provider` a `Broadcast receiver`. Poslední tři komponenty není třeba dále rozebírat, protože nejsou pro tuto práci důležité. Výjimku tvoří samozřejmě aktivita, která je zcela zásadní pro každou aplikací s grafickým rozhraním. Podrobnosti o základních komponentách systému Android jsou k nalezení v oficiální příručce[1].

Aktivita (`Activity`) odpovídá obvykle jedné obrazovce a obsahuje prvky uživatelského rozhraní. Rozložení prvků definuje soubor ve formátu XML, případně je možné tyto prvky definovat či měnit dynamicky v kódu aktivity. Jedna aktivita však může obsahovat více fragmentů a tím zobrazovat i více různých obrazovek. Oproti jiným třídám frameworku právě aktivita a fragment mají specifický životní cyklus, který má své výhody ale často způsobuje i řadu problémů. Základní diagram životního cyklu aktivity je již velmi dobře znám, ovšem

[1] https://developer.android.com/guide/components/fundamentals.html

celá problematika může být značně složitější a to při různých situacích interakce aktivit s fragmenty. Článek Jose Alcérreca se věnuje právě těmto složitějším případům[2]

Stěžejní třídou frameworku je `Context` (česky kontext), která dědí přímo z třídy `Object`. Je to globální rozhraní pro přístup k aplikačním a systémovým zdrojům a třídám. Mnoho operací vyžaduje právě kontext. Nepřímým potomkem této třídy je výše zmíněná aktivita, ale také například třída `Application`[3] udržující globální stav aplikace. Častou chybou je předpoklad, že tyto třídy za běhu aplikace musí být vždy definované (hodnota není `null`). Kontext získaný z aktivity se řídí životním cyklem aktivity a podle toho také současně s aktivitou může zaniknout. Naproti tomu kontext aplikace (možno získat metodou `getApplicationContext()`) není závislý na životním cyklu žádné aktivity, ale celé aplikace, i tato metoda tedy může vrátit `null`. Systém Android je totiž navržen tak, aby mohl aplikace ukončovat podle potřeby, například při nedostatečné volné operační paměti. Pro rozhodování o ukončení procesů je využívána hierarchie důležitosti. Oficiální příručka zmiňuje logiku, podle které systém s procesy aplikací pracuje a případně je ukončuje[4].

Důležitou třídou v aplikaci je také `Intent`. Tato struktura slouží k uložení dat a provedení akce, jako je typicky start nové aktivity, případně operace se službami (`Service`). Je to tedy způsob, kterým se propojují aktivity a předávají mezi nimi data. `Intent` umožňuje předávat základní datové typy, ovšem komplikací může být předávání objektů. To lze vyřešit implementováním rozhraní `Serializable` nebo `Parcelable`.

4.2 Grafické rozhraní

Pravidla pro umístění jednotlivých prvků na obrazovku jsou popsána ve formátu XML. Tyto soubory jsou umístěny v adresáři `"res/layout"`. Jeden soubor může definovat rozložení celé aktivity, nebo jen jednoho znovupoužitelného prvku. Základní třídou pro grafický prvek je `View`. Od této třídy jsou dále odvozeny všechny ostatní grafické prvky.

Výčet několika základních grafických prvků:

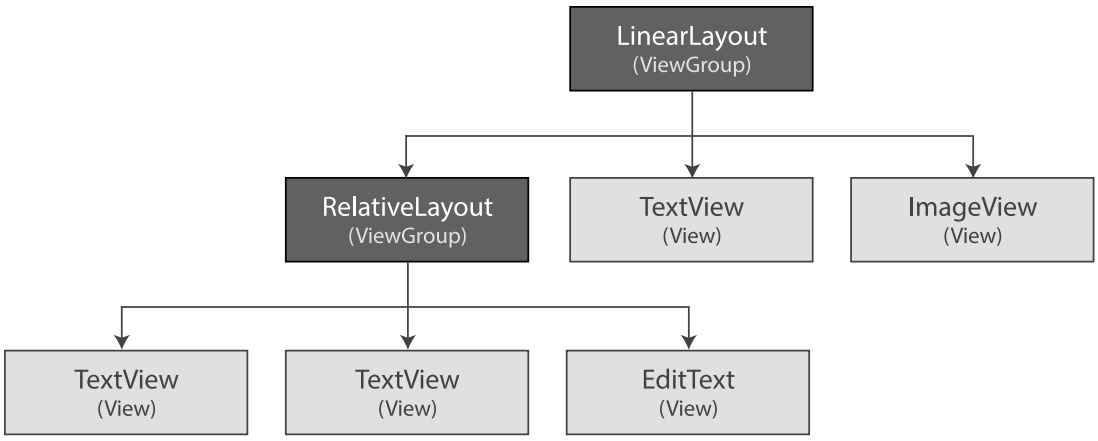
- View – zobrazuje pouze plochu s nastavenými rozměry a barvou pozadí
- TextView – umožňuje zobrazovat texty a nastavit jejich formát
- ImageView – zobrazuje obrázek ve formátu Bitmap nebo Drawable[5]
- Button – tlačítko pro jednoduchou interakci
- EditText – pole pro zadávání nebo úpravu textu
- ViewGroup – kontejner obsahující další prvky uživatelského rozhraní (View) a určující jejich rozložení

[2] https://medium.com/@JoseAlcerreca/the-android-lifecycle-cheat-sheet-part-iii-fragments-afc87d4f37fd

[3] https://guides.codepath.com/android/Understanding-the-Android-Application-Class

[4] https://developer.android.com/guide/components/activities/process-lifecycle.html

[5] Drawable je koncept pro definování grafiky pro zobrazení. Může se jednat o bitmapový soubor nebo XML s různými formáty jako Layer List, State List, Clip Drawable atd. Více informací je možné dohledat v dokumentaci systému Android, viz https://developer.android.com/guide/topics/resources/drawable-resource.

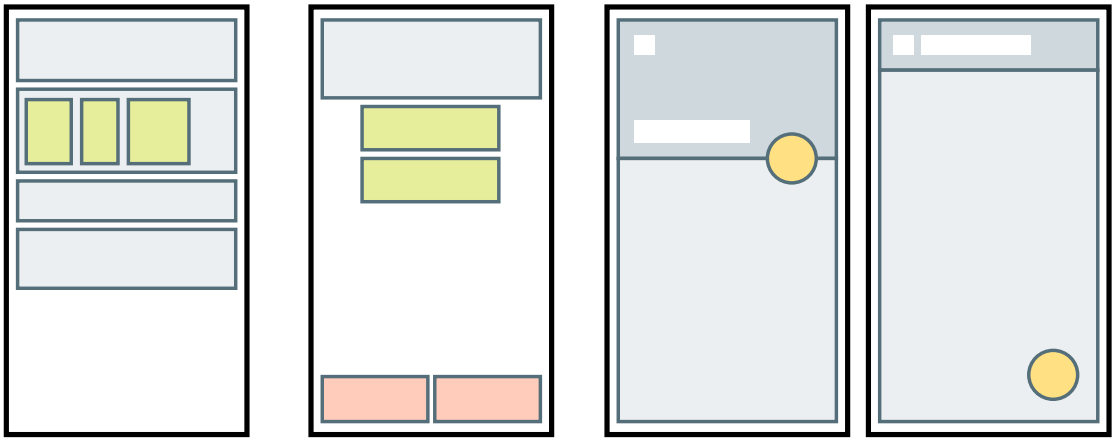


Obrázek 4.1: Ukázka jednoduchého rozložení aktivity zobrazeného jako strom. Hierarchie je tvořena třídami odvozenými od `View` a `ViewGroup`.

4.2.1 Kontejnery grafických prvků

Třída `ViewGroup` je abstraktní, takže se nevyužívá přímo. Jsou na ní postaveny konkrétní kontejnery pro různé typy rozložení. Více o rozvržení grafického rozhraní obsahuje část Layouts v dokumentaci systému Android[6]. Základním rozložením je `LinearLayout`, které vykresluje grafické prvky za sebou ve vertikálním nebo horizontálním směru. Je tedy jednoduché a málo náročné na výkon. Pro složitější rozložení prvků je možno využít více zanořených kontejnerů `LinearLayout` nebo třídu `RelativeLayout`. Ta umožňuje specifikovat pozici vnořených objektů buďto relativně k sobě, nebo ve vztahu ke kontejneru. Dalším využívaným kontejnerem je `FrameLayout`, který je vhodný pro rozmístění objektů na plochu nezávisle na sobě. Doplňuje vlastnost `"layout_gravity"`, kterou se určuje zarovnání uvnitř kontejneru. Zajímavým novějším kontejnerem je `CoordinatorLayout`. Ten rozšiřuje `FrameLayout` o tzv. koncept `Behavior` (česky chování)[7]. Tento koncept ovlivňuje zachytávání veškerých vlastností a interakcí. Tím usnadňuje implementaci chování definovaného stylem Material design. Typické je propojení s grafickými prvky `AppBar` nebo `FloatingActionButton`, jak znázorňuje obrázek 4.2.

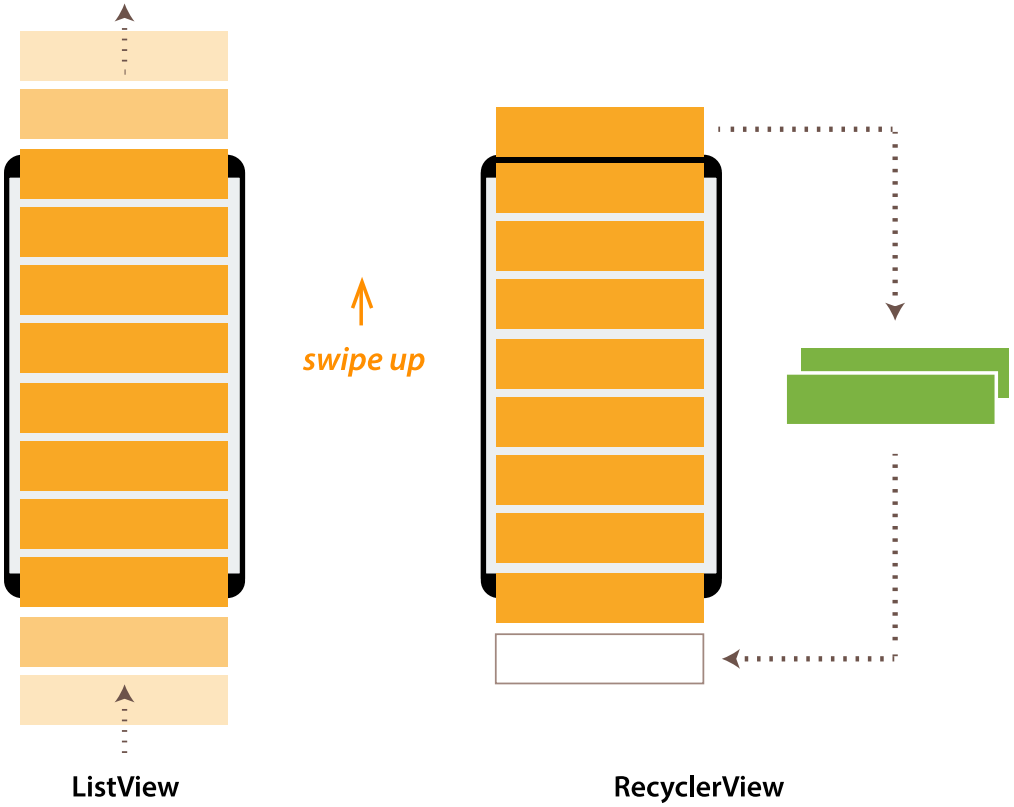
[6] https://developer.android.com/guide/topics/ui/declaring-layout
[7] https://medium.com/google-developers/layouts-attributes-and-you-9e5a4b4fe32c



Obrázek 4.2: Ukázka typického použití vybraných rozvržení. Vlevo LinearLayout s vertikální orientací obsahuje totožný kontejner ovšem s orientací horizontální. Uprostřed RelativeLayout s prvky přichycenými k okrajům kontejneru a také prvky vzájemně přichycenými k sobě. Vpravo potom ukázka kontejneru CoordinatorLayout s prvkem AppBar a tlačítkem FloatingActionButton. Pro lepší pochopení je vyobrazen ve dvou stavech, jaké mohou nastat při vertikálním posunu zobrazení.

Samostatnou skupinou jsou kontejnery používané pro delší až teoreticky nekonečné seznamy. Dříve se využíval `ListView` nebo `GridView`. Dnes je již standardem kontejner `RecyclerView`[8]. Zásadní vlastností je recyklace grafických prvků a využití třídy `ViewHolder` uchovávající načtený grafický prvek[9]. Ve skutečnosti se tedy nejedená o kontejner s vykreslenými všemi prvky, které obsahuje. V danou chvíli jsou vykresleny jen ty, které jsou potřeba. Při posunu zobrazení je zkontrolováno, jestli může být využit již existující prvek pro zobrazení nového. V takovém případě je tento prvek přidán na nové místo (recyklován) a pomocí funkce `onBindViewHolder()` upraven jeho obsah. To ovlivňuje náročnost vykreslování aplikace a také paměťovou náročnost. Rozdílné chování oproti `ListView` je demonstrováno obrázkem 4.3. Samotné rozložení prvků uvnitř tohoto kontejneru je určeno nastavením třídy `LayoutManager`. Základními možnostmi jsou odvozené třídy `LinearLayoutManager` nebo `GridLayoutManager`. Nebo je možné využít rozšiřující knihovnu, či námi vytvořenou třídu. Pro předávání dat kontejneru se využívá Adapter. Tato třída obsahuje úplný seznam objektů, které budou využity pro zobrazení v `RecyclerView` a obsahuje další metody pro práci s těmito daty. Aktuálně požadovaný objekt adaptér předá třídě `ViewHolder` pro vytvoření zobrazení.

[8] https://developer.android.com/reference/android/support/v7/widget/RecyclerView
[9] https://medium.com/mindorks/diffutils-improving-performance-of-recyclerview-102b254a9e4a



Obrázek 4.3: `ListView` vytváří pro každou položku nový grafický prvek (`View`). Alespoň pokud není jeho chování upraveno s využitím konceptu "View Holder". U kontejneru `RecyclerView` je již využití tohoto konceptu povinné a využívá se k tomu třída `RecyclerView.ViewHolder`. Tím je vždy zajištěna recyklace grafických prvků.

4.3 Animace

V systému Android existuje mnoho možností jak provést animaci. Systém vývojáře nijak neomezuje, a tak je výběr konkrétních metod animací čistě na vývojáři. To ovšem neznamená, že všechny jsou vhodné a všechna jejich nastavení budou v každém případě fungovat. Pro komplexnější práci s animacemi je tedy velmi důležité správné použití. [9]

4.3.1 Animace grafických prvků uživatelského rozhraní

`View Animation` je nejstarším typem. Umožňuje animovat pouze 4 vlastnosti, kterými jsou průhlednost (Alpha), zvětšení (Scale), rotace (Rotation) a posun (Translate). Také je možné tyto animace spojovat pomocí AnimationSet. Vytvořit animaci typu `View Animation` je možné jak v XML, tak přímo v kódu jazyka Java. Častou chybou může být chybějící hodnota `"duration"` (česky trvání), která sice není vyžadována, ale bez ní bude délka animace nula milisekund [9], proto k animování nedojde. Velmi nápomocnou vlastností v mnoha případech je `"fillAfter"`, která způsobí zachování koncového stavu animace.

Od verze systému Android 3.0 je možné využívat také pokročilejší způsob `Property Animation`. Tato metoda obsahuje několik přednastavených vlastností. Ve výchozím stavu je již trvání animace nastaveno na 300 milisekund a chování ekvivalentní s vlastností

`"fillAfter"` předchozího typu animace. Kromě toho je také nastavena výchozí třída pro interpolaci (`Interpolator`) na `AccelerateDecelerateInterpolator`. Jednou z možností, jak provést tuto animaci je použití třídy `ValueAnimator`, která pro animování používá plynulou změnu primitivních datových typů jako int nebo float. Změnu této hodnoty je možno odchytávat vnořenou třídou `AnimatorUpdateListener`. S každým voláním její funkce `onAnimationUpdate()` můžeme provést téměř libovolnou operaci s aktuální číselnou hodnotou. Typicky je tedy provedena změna nějaké vlastnosti grafického objektu, čímž dojde k animování. Pokud je potřeba animovat pouze jednu vlastnost jednoho objektu, je jednodušší použít třídu `ObjectAnimator`. Tato třída nevolá opakovaně žádnou funkci. Vše potřebné se nastaví parametry metody `ofFloat()` (respektive `ofInt()`) a třída se již o průběh animování vybraného objektu postará sama.

Další metodou je `View Property Animation`. V mnoha případech se jedná o nejlépe použitelnou metodou. Použití je podobné jako u třídy `ObjectAnimator`, ale syntaxe je jednodušší. Současně tento způsob umožňuje více možností, jako snadné animování více vlastností současně a nastavení více různých interpolací. Animované hodnoty mohou být nastaveny absolutně nebo relativně k aktuálním. Tuto metodu je možné využít jednoduchým voláním metody `.animate()` nad libovolným objektem třídy `View`. Jednoduchou syntaxi demonstruje ukázka kódu 4.1. Ovšem vlastnosti, které je možno tímto způsobem animovat jsou omezeny.

```
1   view.animate()
2      .alpha(0f)
3      .scaleY(0f)
4      .duration(250)
5      .start();
```

Výpis 4.1: Definování a spuštění animace typu `View Property Animaton` nad grafickým prvkem. Ukázka provede zprůhlednění a transformaci výšky.

4.3.2 Animace grafických podkladů

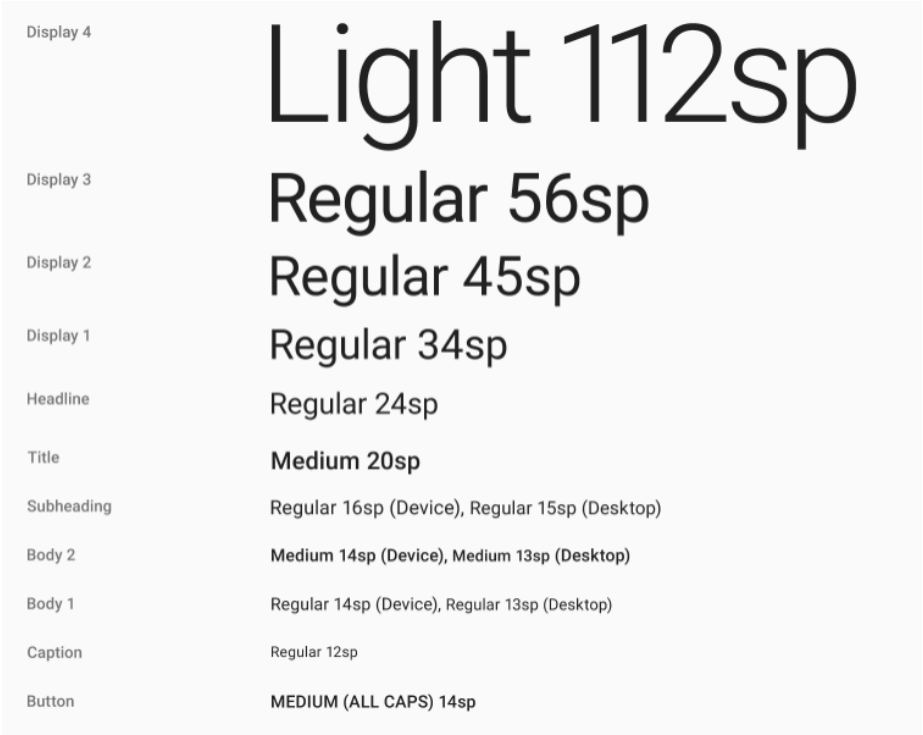
Zcela odlišným typem od předchozích je `Drawable Animation`. Tato animace není aplikovaná na prvek grafického uživatelského rozhraní (`View`), ale přímo na grafický podklad (`Drawable`). Jedná se o jednoduchou změnu obrázku po snímcích. Každý snímek má definovanou svou grafickou podobu (libovolný statický grafický podklad – `Drawable`) a také trvání. Výsledek je tedy spíše přirovnatelný k videu či animacím formátu GIF.

Další možností jak animovat objekty `Drawable` je `Animated Vector Drawable`. U tohoto způsobu už dochází k reálné plynulé animaci. Je ji ale možné aplikovat pouze na vektorové grafické podklady. Podobně jako předchozí typ je i tento definován v XML souboru v adresáři určeném pro objekty `Drawable`.

4.4 Material Design

Material design je designový jazyk představený společností Google v roce 2014 spolu s verzí Android 5.0. Google připravil webovou stránku s kompletní specifikací[10] a současně také knihovnu pro podporu prvků Material designu u starších verzí systému Android. Celá specifikace a sada doporučení vychází ze čtyř tzv. metafor reálného světa. První je digitální papír, což má být plocha v aplikaci připomínající reálný papír s tloušťkou 1 dp[11], ovšem s rozšířenými možnostmi jako je transformace, dělení a spojování. Druhá metafora je digitální inkoust, a tím je myšleno vše co se nachází na digitálním papíře, tedy texty i obrázky. Dalšími metaforami jsou smysluplné animace a adaptivní design.

Pro každou z těchto čtyř metafor jsou definována doporučení, jak s nimi pracovat. Zásadními vlastnostmi, ze kterých ale vše vychází jsou světlo, stín a hloubka. Přesto, že uživatel vidí prvky aplikace vždy jen seshora, hloubka je znázorněna různě ostrým a širokým stínem. Uživatel tak vidí vždy jen prvek, který je nejvýše a současně podle stínu může vytušit jak vysoko oproti ostatním je. Interakce taktéž probíhá pouze s nejvýše položenými prvky.



Obrázek 4.4: Standardním fontem Material designu je Roboto. Pro různé použití jsou navíc definovány velikosti a řezy písma. Převzato z: https://material.io/guidelines/

[10] https://material.io/guidelines/

[11] DP nebo DIP (Density-independent Pixels) je abstraktní jednotkou používanou v systému Android. Na rozdíl od PX neudává počet reálných pixelů ale jejich přepočet podle fyzické hustoty pixelů. DP a PX mají stejnou velikost na obrazovce s hustotou 160 dpi.

4.5 Architektury aplikací

Architektura aplikace pro systém Android není nijak striktně určena, je tedy možné aplikaci navrhnout více způsoby. Definování a dodržování určitého architektonického vzoru (architectural pattern) je vhodné pro udržení přehledného a funkčního kódu. Jednotlivé vzory také ovlivňují, jak dobře je aplikace testovatelná, nebo jaká je míra znovupoužitelnosti kódu. Běžnou a přesto zásadní chybou je psaní veškerého kódu do aktivit či fragmentů [3]. Dále jsem provedl stručný souhrn informací o třech nejčastěji využívaných architekturách, které blíže popisuje Kumar [11].

4.5.1 Model-View-Controller

`Model-View-Controller` je nejstarším architektonickým vzorem. Hlavní vrstvou je řadič (`Controller`), který má na starost použití správného modelu a zobrazení (`View`). Vrstva `View` (XML layout) se stará o zobrazování uživatelského rozhraní a také doručování uživatelských interakcí do řadiče. Model (Java) se stará o zisk dat a byznys logiku. Aktualizace modelu jsou předávány zobrazení. Řadič provádí manipulaci s modelem, případně i zpracování dat získaných z modelu a předávání vrstvě `View`. Řadič je ale obvykle přímo součástí aktivity nebo fragmentu. V případě, že je řadič oddělen do samostatné třídy, je mu předávána reference na tuto aktivitu. Nevýhodou je závislost řadiče i modelu na vrstvě `View`. Řadič obvykle provádí logiku a současně využívá i Android API. To má za následek horší testovatelnost.

4.5.2 Model-View-Presenter

V architektuře `Model-View-Presenter` je zobrazení (vrstva `View`) tvořeno celou aktivitou či fragmentem. Opět se tedy stará o zobrazování uživatelského rozhraní, ale také přijímá uživatelské interakce. Vrstva `Presenter` je samostatná třída komunikující s modelem. Není přímo závislá na zobrazení, ale slouží jen jako rozhraní. Zobrazení nikdy nekomunikuje přímo s modelem, ale využívá k tomu rozhraní vrstvy `Presenter`. Třídy vrstvy `Presenter` a `Model` by neměly být závislé na Android API. Při dodržení těchto pravidel je výsledný kód lépe testovatelný než u architektury MVC.

4.5.3 Model-View-ViewModel

`Model-View-ViewModel` obsahuje vrstvu `ViewModel` získávající data z modelu. Nad těmito daty provádí logiku a následně ji poskytuje komukoli, kdo by daná data potřeboval. Vrstva `View` (tvořena aktivitou či fragmentem) tak může být právě tím, kdo na tyto data ze třídy `ViewModel` naslouchá a následně je zobrazuje uživateli. Vrstvy `Model` i `ViewModel` jsou tedy nezávislé na Android API i vrstvě zobrazení. Data z vrstvy `ViewModel` tak můžou být kromě zobrazování použita i pro snadné testování. Pro tuto architekturu je vhodné využít knihovnu Data Binding systému Android. Ta umožňuje vytvořit pozorované proměnné (`ObservableField`). Díky tomu lze předávat aktivitě (a dalším třídám) aktualizace stavu bez přímé reference.

Další možností je také využít novější soubor knihoven Architecture Components. Cílem bylo zjednodušení implementace a zvýšení robustnosti aplikací postavených na principu MVVM. Jedná se o sadu komponent, které je možno využít i jako samostatné knihovny. Více je o tomto tématu uvedeno v kapitole 5.4.

Kapitola 5

Použité knihovny

V projektu jsem využil základní knihovny systému Android pro zpětnou kompatibilitu jako `App Compat Library` a `Design Support Library`. Využíval jsem také knihovnu Retrolambda zpřístupňující prvky jazyka Java 8, jako je například lambda výraz (lambda expression). Tuto knihovnu už není nutno využívat, protože od verze Android Studio 3.0 je možné používat jazyk Java 8 bez doplňujících knihoven.

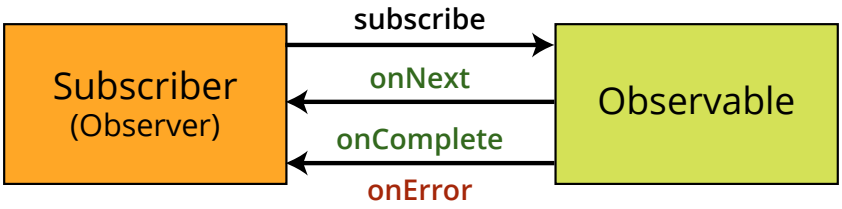
5.1 Data Binding

Data Binding[1] je knihovna systému Android pro usnadnění synchronizace uživatelského rozhraní s logikou. K tomu je využíván deklarativní layout (česky rozložení). Soubor s rozložením tedy obsahuje navíc propojení prvků grafického rozhraní s daty. Pro každou aktivitu je automaticky vygenerována třída, která bude provádět námi deklarovanou práci s daty (pro aktivitu s rozložením activity_main.xml vygeneruje třídu ActivityMainBinding). Takovýto layout dokáže dynamicky měnit své parametry, typicky hodnoty textových polí, podle stavu přiřazeného objektu. Současně lze díky této knihovně přistupovat v kódu aktivity snadněji k prvkům grafického rozhraní (View). Místo běžného zápisu `layout.findViewById(R.id.textovy_prvek)` je možné použít `binding.textovyPrvek`. Data Binding je základem pro architekturu MVVM. ViewModel neobsahuje referenci na vrstvu View, ale naopak vrstva View využívající Data Binding může propojit grafický prvek s daty objektu ViewModel.

5.2 Retrofit

Retrofit 2[2] je knihovna vytvářející HTTP klienta vhodného pro komunikaci s REST API. Všechny dotazy jsou jednoduše deklarovány formou rozhraní (Interface) jazyka Java. Jsou podporovány běžné metody GET, POST, PUT, DELETE či HEAD. Pro serializaci a deserializaci dat ve formátu JSON navíc využívám knihovnu GSON[3] od společnosti Google. Retrofit tedy zjednodušuje tvorbu dotazů a příjem odpovědí. Pro samotné sestavení a odesílání HTTP dotazů ale využívá knihovnu OkHttp[4]. Ta navíc obsahuje funkcionalitu pro práci s mezipamětí a zotavení z chyb.

[1] https://developer.android.com/topic/libraries/data-binding/index.html
[2] http://square.github.io/retrofit/
[3] https://github.com/google/gson
[4] http://square.github.io/okhttp/



Obrázek 5.1: Návrhový vzor Pozorovatel implementovaný metodami knihovny RxJava 2. Pozorovatel (Observer) neboli odběratel (Subscriber) na levé straně se zapíše k odběru aktualizací. Ty následně dostává s každým voláním metody `onNext()`, případně informaci o ukončení či chybě metodami `onComplete()` a `onError()`.

5.3 RxJava

Pro práci s datovými toky a organizaci asynchronních procesů jsem využil knihovnu RxJava 2[5]. Ta usnadňuje vytváření programů založených na událostech (event base) a využívá návrhový vzor Pozorovatel (Observer). Tento návrhový vzor je vhodný v případě, kdy je potřeba nějaké objekty informovat o změně stavu pozorovaného objektu. Principu, kterým tato knihovna rozšiřuje možnosti funkcionálního programování se také říká reaktivní programování (Functional Reactive Programming) [8]. Knihovna RxJava spolu s rozšířením RxAndroid a adaptérem pro knihovnu Retrofit umožňuje provádět komunikaci s REST API asynchronně v novém vlákně. Přiřazený pozorovatel (Observer) tak naslouchá na výsledek dotazu a stará se o následné zpracování vrácené odpovědi, případně chyby. Konkrétní metody, které využívá RxJava jsou uvedeny v obrázku 5.1.

5.4 Architecture components

Architecture Componets [2], někdy také nazýváno Lifecycle Architecture Components, je sada knihoven představena společností Google na konferenci Google I/O v roce 2017. Tyto knihovny upravují celkovou architekturu aplikace pomocí nových objektů jako LiveData, ViewModel, ViewModelProviders, Lifecycle, LifecycleObserver a řadou dalších. Cílem bylo zlepšit práci s životními cykly (Lifecycle) a jak uvádí společnost Google, udělat aplikaci více robustní, testovatelnou a udržovatelnou. Tyto rozšiřující komponenty upravují způsob implementace architektury MVVM a zlepšují udržovatelnost zejména při práci s tzv. živými daty[6].

Společnost Google ovšem opět představila novinky na poslední konferenci Google I/O 8. 5. 2018. Došlo k aktualizaci tohoto souboru knihoven a sjednocení pod názvem Android Jetpack[7]. Tyto změny ale tato práce nebude popisovat.

[5] https://github.com/ReactiveX/RxJava

[6] https://medium.com/exploring-android/exploring-the-new-android-architecture-components-c33b15d89c23

[7] https://android.jlelse.eu/what-is-android-jetpack-737095e88161

5.5 Glide

Knihovna Glide[8] se stará o načtení bitmapového obrazu do elementu `ImageView`. Je vhodná pro získání obrazu z URL adresy a umožňuje snadné využití paměti cache. Dokáže automaticky přizpůsobit rozlišení načteného obrazu velikosti cílového grafického prvku. Nebo je možné nastavit přesné rozlišení a transformace. Při používání většího množství obrázků tedy dokáže snížit datovou i výpočetní náročnost.

5.6 Simple Stateful Layout

Stateful Layout[9] je knihovna navržená pro zjednodušení práce se stavem rozložení (layout). Využívám variantu "Simple", protože mi svou funkcionalitou dostačuje. Knihovna přidává nový grafický prvek (View) s názvem `SimpleStatefulLayout`, který obsahuje volitelné atributy `app:offlineText`, `app:emptyText`, `app:state` a další. Atributy je možné nastavit chování a vzhled tohoto prvku pro různé stavy. Tyto stavy poté lze snadno přepínat voláním funkcí `showProgress()`, `showContent()`, `showOffline()` nebo například `setState()`.

5.7 Flow Layout

Android obsahuje několik základních tříd typu ViewGroup, ty se starají o rozvržení grafických prvků (View) v uživatelském prostředí (layout). Žádná ovšem nedokáže vytvořit automaticky zalamovaný víceřádkový layout. Toto je možno buďto simulovat dynamickým vytvářením objektu LinearLayout pro každý nový řádek, nebo je nutno použít externí knihovnu. Takovou knihovnou je Flow Layout[10]. Výsledkem může být chování totožné se známým atributem jazyka CSS `display: inline-block`.

5.8 AndroidUtilCode

Pro využití různých pomocných metod se často používají nástrojové třídy (utility class). Knihovna AndroidUtilCode obsahuje celou řadu pomocných nástrojů pro práci s aktivitami, šifrováním, internetovým připojením, službami systému nebo textovými řetězci. Například metoda `NetworkUtils.isConnected()` vrací logickou hodnotu určující, zda je zařízení připojeno k internetu.

5.9 Overscroll Decor

Pro možnost zavřít aktivitu přetažením shora nebo zespodu jsem využil knihovnu Overscroll Decor[11]. Tato knihovna přidává zpětné volání (callback) reagující na tažení prvkem `ScrollView` za jeho běžný rozsah. Současně dojde k vizuálnímu efektu posunu celého `ScrollView` a odkrytí elementů pod ním. Podle hodnoty `offset` udávající vzdálenost přetažení je možné nastavit různé reakce.

[8] https://github.com/bumptech/glide
[9] https://github.com/jakubkinst/Android-StatefulLayout
[10] https://github.com/ApmeM/android-flowlayout
[11] https://github.com/EverythingMe/overscroll-decor

5.10 Další použité kódy

Protože řadu věcí Android SDK přímo nepodporuje, využil jsem dále několik zdrojových kódů, které autoři umístili na web. Jedním takovým problémem je špatně fungující nastavení mezer mezi prvky v `RecyclerView`. Proto jsem použil třídu pojmenovanou `GridSpacingItemDecoration`[12] od vývojáře Ailurus (ailurus@foxmail.com).

Další třídou je `TouchImageView`[13], která rozšiřuje prvek `ImageView`. Doplňuje uživatelská gesta jako "pinch-to-zoom", reakce na dvojklik apod. Tuto třídu jsem dále rozšířil o callback, na který dokáže reagovat aktivita a možnost posunout prstem obrázek nahoru či dolů, čímž lze detail obrázku zavřít podobně jako to u prvku `ScrollView` umožnila knihovna Overscroll Decor.

`RoundedCornerLayout` je třída rozšiřující `RelativeLayout` o zaoblené rohy. Funguje na principu simulování masky. Rohy jsou překryty jednotnou barvou, která musí být nastavena i pro pozadí aktivity. Tuto třídu jsem upravil pro konkrétní použití v aplikaci. Zaoblení rohů jsem nastavil na 10dp (převedeno na počet pixelů pomocí třídy `ImpresUtil`, která je popsána v kapitole 7.5).

[12] https://gist.github.com/liangzhitao/e57df3c3232ee446d464
[13] https://github.com/MikeOrtiz/TouchImageView

Kapitola 6

Návrh aplikace Impres

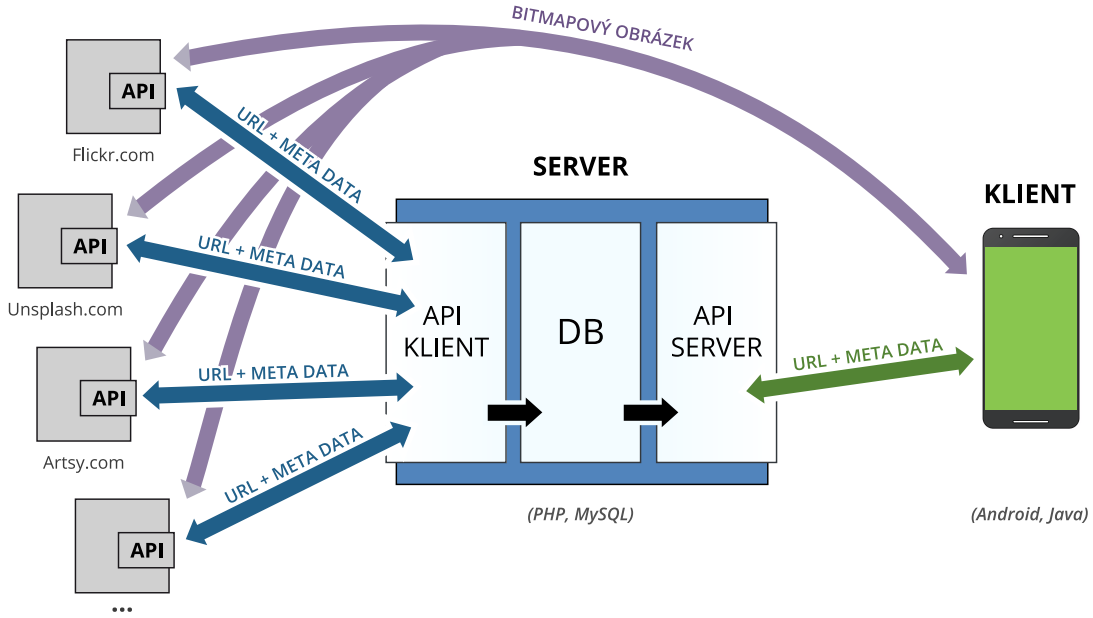
Na základě motivace z kapitoly 2 a analýzy existujících řešení v kapitole 3 jsem si ujasnil cíle mobilní aplikace Impres. Aplikace je určena zejména pro ty, kteří inspiraci potřebují nebo chtějí získávat více, ale i pro ty, kteří již jsou zvyklí konzumovat vizuální umění jinými způsoby. Proto jsem provedl několik konzultací s lidmi této cílové skupiny. Dále jsem pracoval na celém kreativním procesu návrhu grafického uživatelského rozhraní.

Mobilní aplikace Impres bude poskytovat vizuální obsah uživateli. Tento obsah bude tvořen fotografiemi a obrázky z více internetových zdrojů. Uživatel by neměl tuto skutečnost téměř vnímat, protože se veškerý obsah bude zobrazovat jednotným způsobem. Na základě požadavků jednotlivých služeb bude pouze v aplikaci uvedena informace o zdroji daného obrázku. Kromě samotného obrazového obsahu bude aplikace využívat další informace. Mezi tyto patří název, klíčová slova, případně i autor obrázku. Klíčová slova budou použita pro doporučování obsahu uživateli. Přesto nebude po uživateli aplikace vyžadováno přihlášení a další dodatečné aktivity, které by snižovaly míru anonymity. Personalizace obsahu bude vycházet z aktivity uživatele v aplikaci, bez potřeby znát jeho identitu. Obrázky bude možno označit jako oblíbené a tím si je přidat do své sbírky.

6.1 Vytyčené cíle aplikace

Rozhodl jsem se jasně sepsat zásadní body, které by aplikace Impres měla splňovat. Některé vycházejí z motivace a základního konceptu, jiné pak reagují na zjištěné dobré i špatné vlastnosti analyzovaných existujících řešení.

- Přehlednost a jednoduchost grafického rozhraní
- Personalizovaný obsah – doporučené obrázky uživateli
- Možnost přidávat obrázky do sbírky oblíbených
- Přizpůsobené rozložení různým obrázkům a jejich poměrům stran
- Gesta pro zvýšení ergonomie aplikace a zrychlení použití
- Animace podporující celkový prožitek při používání
- Anonymita – nevyžadování osobních údajů a kontaktních informací
- Offline režim – možnost omezeného použití i bez přístupu k internetu



Obrázek 6.1: Komunikace mezi poskytovateli obsahu (šedě), serverovou aplikací (modře) a klientskou mobilní aplikací (zeleně). Komunikace probíhá obousměrně, protože je typu dotaz–odpověď. Strukturovaná data získaná z veřejných REST API poskytovatelů obsahu zpracovává server, samotný soubor s bitmapovým obrazem je ale přenášen přímo (fialově). Uvedené názvy poskytovatelů obsahu mohou být pouze ilustrativní.

6.2 Komunikace se serverem

Protože se bude jednat o mobilní aplikaci pracující s internetovým obsahem, bude velmi důležitá komunikace. V tomto případě se jedná o obsah z více zdrojů, teoreticky neomezeného počtu. Aplikace však bude vše zobrazovat jednotným způsobem. Je tedy potřeba zajistit, aby s tím neměla problémy. To je vyřešeno vlastní serverovou aplikací, která se o zisk a zpracování dat postará. Server bude manuálně nebo automaticky získávat data z ostatních služeb, čímž bude mít i roli klienta. Tato data sjednotí a uloží do databáze s informací o původu obsahu.

Data budou serverem poskytována v jednotném datovém formátu právě mobilní aplikaci Impres. Proto je nutné celou tuto komunikaci vhodně navrhnout a definovat rozhraní REST API. Jak uvádí [5], služba se považuje za RESTful[1], když splňuje tato omezení: jednotné rozhraní, princip klient–server, bezstavovost, využití dočasné paměti a další. Cílem správně navržené komunikace je určitá míra abstrakce. To znamená, že z pohledu mobilní aplikace je server jen sada jasně popsaných koncových bodů a předem definovaná struktura zasílaných dat. Z pohledu serverové aplikace naopak není vůbec důležité, jak mobilní aplikace s daty pracuje, server pouze vystaví data v této definované podobě a o více se nestará. Podobně je to s konkrétní použitou technologií. Kdy klientská aplikace je naprogramována v jazyce Java pro platformu Android a serverová v jazyce PHP. Když v budoucnu dojde k změně technologie jedné z aplikací, druhou to nijak neovlivní. Bezstavovost znamená, že aktuální stav klientské aplikace není ukládán na server. Dočasná paměť bude využívána na straně

[1]RESTful je termín, kterým jsou označovány služby používající REST API.

Obrázek 6.2: Logo aplikace Impres. Na základě slepých testů, které jsem provedl na dalších osobách, jsem zjistil, že předchozí varianty loga nebyly dobře rozeznatelné. Hlavním problémem byla identifikace písmena "m". Proto jsem prováděl testování a drobné úpravy, ze kterých vzniklo finální logo.

klienta. Protože služba nepracuje s osobními údaji ani kontaktními informacemi a veškerý obsah je volně dostupný, zabezpečení přenosu nebylo nutno řešit. Celou tuto komunikaci znázorňuje zjednodušené schéma na obrázku 6.1.

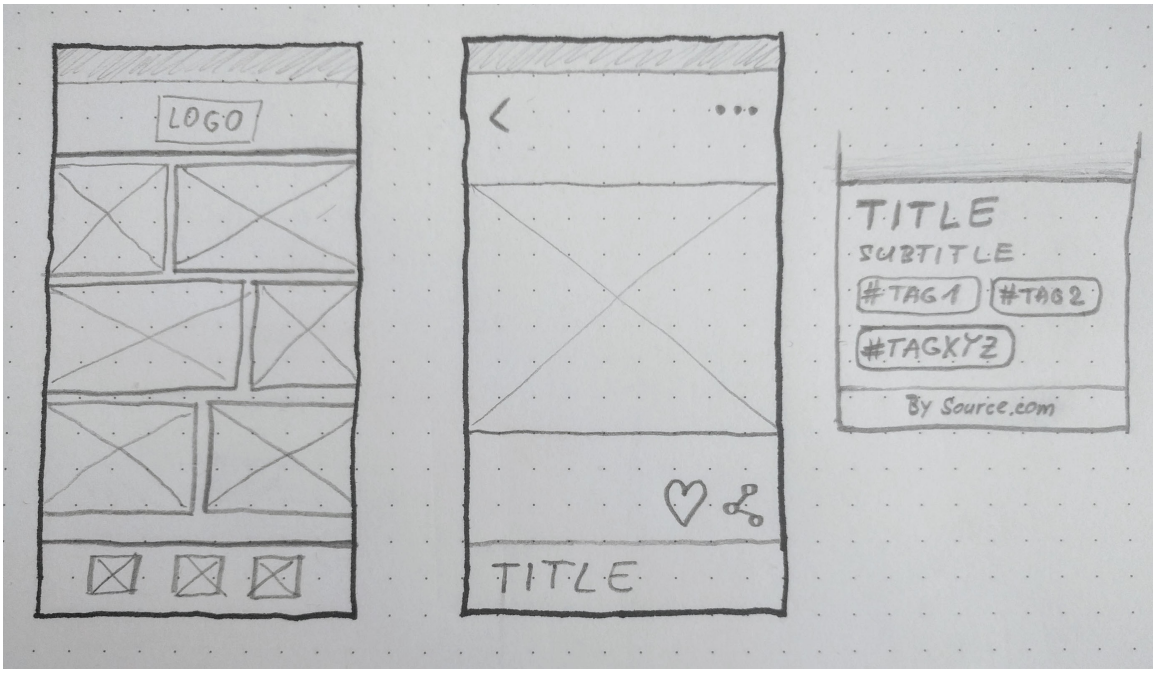
6.3 Návrh uživatelského rozhraní

Začal jsem tvorbou náčrtů (sketch) uživatelského prostředí. Tyto nemusí být přesné, ale jsou vhodné zejména pro ujasnění hrubého rozmístění prvků. Obrázek 6.3 obsahuje náčrt dvou obrazovek (na vhodném tečkovaném papíru). Dále jsem provedl high-fidelity prototypování (prototyping) obrazovek[2]. Níže jsou popsány důležité obrazovky. Kromě návrhu hlavních obrazovek jsem navrhl také logo aplikace 6.2.

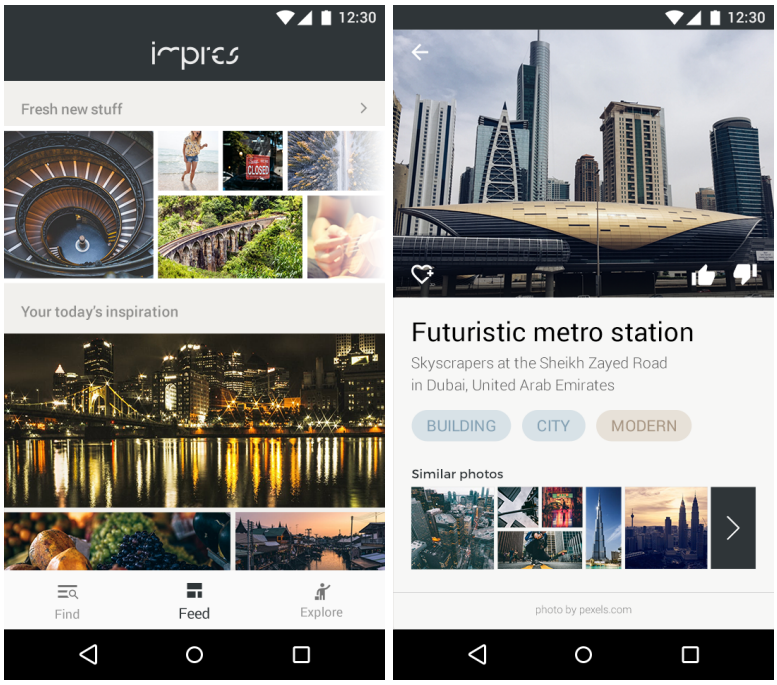
6.3.1 Hlavní obrazovka

Hlavní obrazovka obsahuje spodní navigační lištu a horní lištu s logem aplikace. Horní lišta se bude automaticky skrývat. Mezi těmito lištami bude seznam obrázků. Obrázky půjde otevřít jednoduchým klepnutím. Spodní lištou bude možné přepínat mezi různými typy seznamů. Těmito typy jsou doporučené obrázky, nejnovější obrázky a sbírka oblíbených. Podle náčrtu jsem následně pro tuto obrazovku připravil statický prototyp 6.4. Ten obsahuje i ukázku možné kombinace obsahu doporučeného s nejnovějším, což nepatří mezi vytyčené cíle.

[2]Prototypování je tvorba neúplných ukázek výsledného produktu, používaná pro analýzy grafického designu, uživatelského zážitku apod. High-fidelity (česky vysoká věrnost) značí prototyp s velmi vysokou přesností detailů. Naopak za Low-fidelity prototyp může být považován i hrubý náčrt.



Obrázek 6.3: Ruční náčrty uživatelského rozhraní. Obrazovka seznamu obrázků, obrazovka detailu a další prvky. Tento návrh je velmi blízký konečnému řešení.



Obrázek 6.4: Prototyp důležitých obrazovek aplikace. Vlevo hlavní obrazovka s výpisem obrázků. Vpravo obrazovka detailu jednoho obrázku. V těchto obrazovkách jsem navrhl vzhled případných funkcí, které v původním návrhu nebyly plánovány, například automatické zobrazení podobných obrázků.

6.3.2 Detail obrázku

Obrazovka s detailem je taktéž vyobrazena na prototypu 6.4. Tato obrazovka bude obsahovat následující:

- Velký obrázek
- Tlačítko zpět
- Tlačítka pro akce s obrázkem – přidat do oblíbených, pozitivní a negativní ohodnocení
- Název obrázku
- Výpis klíčových slov – pro otevření obrazovky daného klíčového slova

6.3.3 Úvodní obrazovky

Při prvním spuštění aplikace se zobrazí obrazovka s představením hlavních funkcí. Ta bude obsahovat obrázky a text. Tuto obrazovku bude možno přeskočit.

Poté se otevře obrazovka se seznamem nejpoužívanějších klíčových slov. Uživatel vybere klepnutím klíčová slova podle osobních preferencí. Zobrazení a potvrzení těchto dat bude vyžadovat internetové připojení. Bez úspěšného odeslání oblíbených témat nebude možné spustit hlavní část aplikace.

6.4 Systém doporučování obsahu

Jednou z funkcí mobilní aplikace bude doporučování neboli personalizace obsahu. Cílem personalizace je poskytovat uživateli pro něj nejvíce relevantní obsah. Systémy doporučování obsahu se využívají ve velké míře u sociálních sítí, reklamních systémů a dalších služeb spadajících pod tzv. e-commerce[3].

Tyto systémy je možno rozdělit do několika kategorií podle způsobu zpracování a použitého typu dat. Podle otevřené encyklopedie Wikipedia.org[4] i Aggarwala [6] mezi základní typy patří tyto:

- Doporučování na základě obsahu (Content-based recommendation) – Na základě preferencí uživatele je ohodnocen jeho vztah k položkám systému. Algoritmus poté vrací položky, které dosáhly nejvyšší pozitivní hodnocení.
- Kolaborativní filtrování (Collaborative filtering) – Tento systém využívá informace o dalších uživatelích systému. Je prováděno porovnávání preferencí aktuálního uživatele s dalšími uživateli. Tato metoda je často využívána u sociálních sítí.
- Hybridní systém – Systém využívající kombinaci více metod. Kombinování může být vážené, což znamená, že je výstup jednotlivých metod zkombinován do výsledného hodnocení podle váhy každé metody. Může se také jednat o přepínání metod, kdy je výsledné hodnocení tvořeno pouze jednou metodou, která je v daném případě nejvhodnější.

[3] E-commerce označuje elektronické podnikání či obchodování. Nejčastějším příkladem jsou internetové obchody. Tento termín blíže popisuje otevřená encyklopedie Wikipedia.org. Dostupné z URL: https://cs.wikipedia.org/wiki/Elektronick%C3%A9_obchodov%C3%A1n%C3%AD

[4] https://en.wikipedia.org/wiki/Recommender_system

V aplikaci Impres bude použit jednoduchý systém doporučování na základě obsahu (Content-based recommendation system). Tento systém bude pracovat s klíčovými slovy obrázků (tag) a preferencemi konkrétního uživatele. Výhodou je, že tento způsob nevyžaduje velké množství uživatelů služby a nebo další informace o nich. Celý systém bude implementován v serverové aplikaci, a proto bude možné provádět jeho úpravy a vylepšení bez zásahu do mobilní aplikace. Pro zachování anonymity nebude vyžadováno přihlášení ani žádné zadávání osobních informací. Pro identifikaci uživatele a následné přiřazování preferencí bude vygenerováno číslo ID. Toto číslo bude uloženo na serveru, ale současně i v mobilním zařízení.

6.4.1 Algoritmus doporučování obsahu služby Impres

Systém pracuje s množinami všech uživatelů (U), obrázků (O), klíčových slov (K) a vztahů uživatele s klíčovým slovem (V). Pro každý obrázek $o \in O$ existuje libovolný počet klíčových slov $k \in K$. Pro každého uživatele $u \in U$ existuje vztah $v \in V$ ke každému klíčovému slovu vyjádřený celým číslem. Nový uživatel má vztah ke všem klíčovým slovům roven nule. Pokud existují pro daného uživatele vztahy ke klíčovým slovům obrázku, je ohodnocení obrázku průměrem těchto vztahů. Ve výpočtu využívám průměr místo součtu, čímž snadno kompenzuji nepoměr v počtu klíčových slov mezi obrázky.

Kroky algoritmu:

1. Získáme N nejnovějších obrázků z množiny O. Tuto množinu nazveme O_n.
2. Nechť pro uživatele u existuje množina V_u vztahů ke všem klíčovým slovům množiny K. Každý vztah $v_u \in V_u$ má tvar uspořádané trojice (u, k, v) kde u je identifikátor uživatele, k klíčové slovo a v vztah vyjádřený celým číslem.
3. Pro každý obrázek o_n z množiny O_n bude vypočítáno ohodnocení $r_{u,o}$ následujícím způsobem. Nechť množinou všech klíčových slov obrázku o_n je K_o. Provedeme výběr pouze těch vztahů V_u, které obsahují klíčové slovo z množiny K_o. Tím získáme množinu všech vztahů daného uživatele ke klíčovým slovům daného obrázku, kterou si nazveme $R_{u,o}$.
4. Výsledné ohodnocení obrázku $r_{u,o}$ je vypočítáno jako průměr všech vztahů množiny $R_{u,o}$.
5. Podle vypočítaného ohodnocení $r_{u,o}$ jsou obrázky seřazeny v sestupném pořadí. Obrázky s nižším než nulovým ohodnocením jsou odfiltrovány.

6.4.2 Získávání dat pro doporučování obsahu

Aby mohl tento algoritmus fungovat, je potřeba od uživatele získávat data o používání aplikace. Protože jedním z cílů aplikace Impres je i anonymita, aplikace nebude vyžadovat osobní údaje a kontaktní informace. Veškerá data vyjadřující osobní preference daného uživatele tedy budou získávána pouze z používání této aplikace. Tyto data budou získávána implicitním i explicitním způsobem.

Explicitně získaná data uživatel přímo zadává. Bude se jednat o úvodní obrazovku aplikace, kde uživatel při prvním spuštění vybere své zájmy. Aplikace tyto data odešle na server, kde dojde ke zpracování. Po přijetí odpovědi bude uživateli následně zpřístupněna

celá aplikace. Nedojde tak ke stavu, kdy by nebylo podle čeho obsah doporučovat. Dalším způsobem bude přímé ohodnocení fotografie pro to určenými tlačítky. Ta budou zadávat pozitivní i negativní hodnocení.

Naopak implicitním způsobem bude aplikace získávat sledováním ostatní aktivity uživatele. Pokud například uživatel klepne na obrázek v seznamu obrázků, dojde k otevření detailu obrázku, ale také k odeslání nízkého pozitivního hodnocení. Podobně také při klepnutí na klíčové slovo u obrázku. Těmito akcemi v aplikaci uživatel přímo nezamýšlí určovat své preference, lze podle nich ale odhadnout, jaká témata ho nejspíše zaujala. Tento způsob hodnocení má však výrazně nižší váhu než přímé explicitní hodnocení.

Kapitola 7

Implementace mobilní aplikace

Hlavní částí celé služby je mobilní aplikace pro operační systém Android. Proto vývoj probíhal v oficiálním vývojovém prostředí Android Studio 3.1.1. Tento program je založen na oblíbeném IntelliJ IDEA od JetBrains. Aktuální verzí systému je Android Oreo 8.1 (API 27) ale minimální podporovaná verze této aplikace je 5.0 (API 21). Některé funkce využívané v aplikaci totiž nejsou podporovány staršími verzemi systému ani knihovnami Support Library pro zpětnou kompatibilitu. Podle přehledu platforem na stránkách Android Developers[1] k 25. dubnu 2018 obsahovalo 84,3 % zařízení alespoň verzi API 21. Protože jsem se rozhodl používat řadu moderních funkcí systému Android, stará zařízení v této verzi nebudou podporována. Tato kapitola bude dále popisovat architekturu a důležité části aplikace.

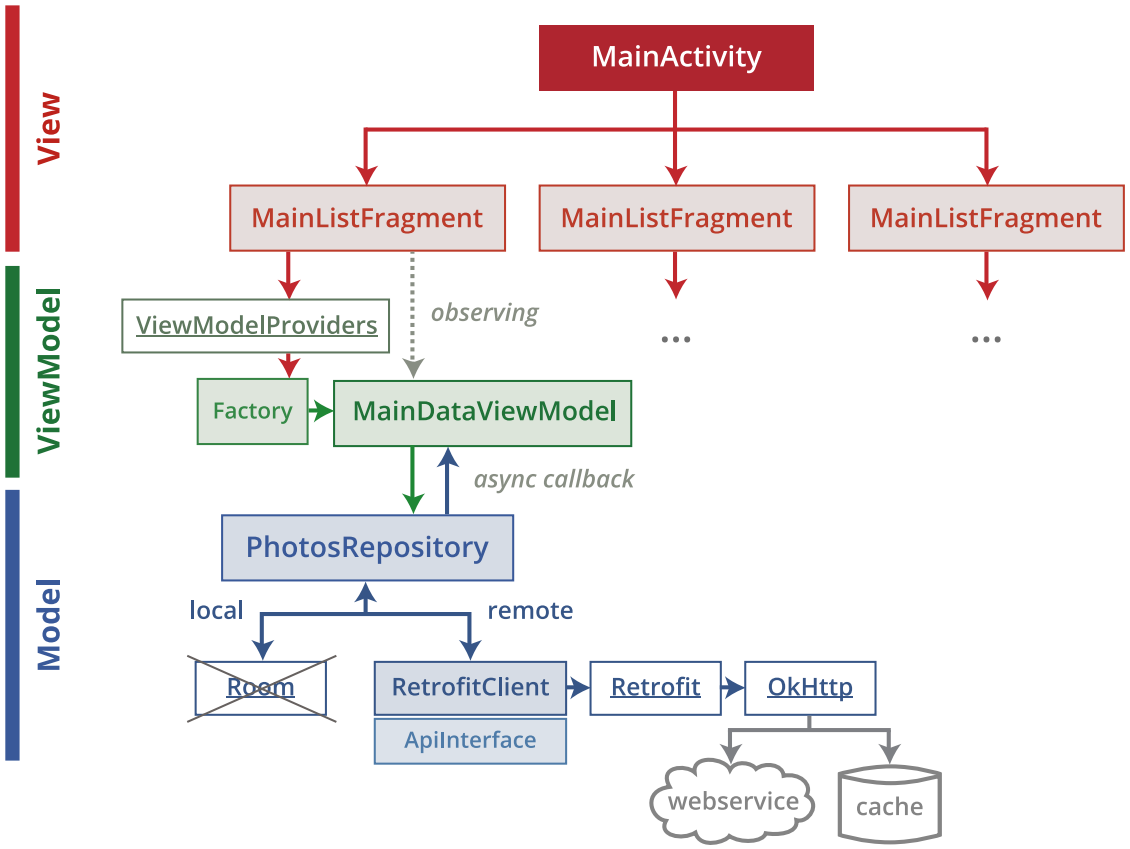
7.1 Architektura

Rozhodl jsem se využít architekturu Model-View-ViewModel spolu s knihovnou Data Binding. Později jsem doplnil i novější knihovnu Architecture Componenents systému Android. Tato architektura je doporučována společností Google. Zvolil jsem si ji také proto, že je vhodná pro výuku novějších knihoven a prvků systému Android [13]. Zásadní vlastností je, že vrstva ViewModel neobsahuje žádnou referenci na vrstvu View, čímž je na ní zcela nezávislá. Tuto architekturu demonstruje na konkrétním příkladu obrázek 7.1. Na tomto diagramu jsou vyobrazeny vztahy mezi třídami, které využívá aktivita `MainActivity` zobrazující hlavní obrazovku se třemi fragmenty. Každý fragment pomocí třídy `ViewModelProviders` z knihovny Architecture Components a třídy `MainDataViewModel.Factory` vytvoří instanci `MainDataViewModel`.

7.1.1 Vrstva Model

Veškeré získávání dat jsem oddělil do samostatné třídy `PhotosRepository`. Tato třída přijímá z vrstvy ViewModel požadavky na konkrétní typy dat a stará se o to, aby byly asynchronně doručeny správnému objektu vrstvy ViewModel. Jejím účelem je zapouzdřit veškerou práci s daty a poskytovat jen výsledná zpracovaná data. Tato data můžeme dělit podle zdroje na vzdálená a lokální. Vzdálená data jsou získávána ze serveru pomocí klienta API. Implementace této části je popsána v kapitole 7.3. Pro lokální data je vhodné využít knihovnu Room. Nicméně ve výsledném řešení nebylo potřeba pracovat s daty tímto způsobem.

[1] https://developer.android.com/about/dashboards/



Obrázek 7.1: Architektura aplikace (zjednodušená) z pohledu aktivity `MainActivity`. Jednotlivé vrstvy jsou odlišeny barevně. Mezi vrstvami View a ViewModel existuje jednostranná reference a předávání dat pomocí knihovny Data Binding a Architecture Components. Komunikace s vrstvou Model probíhá asynchronně díky knihovně RxJava. Třídy s podtrženým názvem jsou z externích knihoven. Lokální úložiště ovládané knihovnou Room ve výsledném řešení není využíváno, zde je znázorněno případné umístění v architektuře. O lokální držení dočasných dat se stará knihovna OkHttp.

Potřebu práce s lokálními daty nahradila dočasná paměť knihovny OkHttp. Při inicializaci této třídy dochází ke kontrole uživatelského ID, které je nutné pro některé dotazy na server. Pokud ještě nebylo vygenerováno (a uloženo do Shared Preferences[2]), `PhotosRepository` se postará o zaslání požadavku na server a získání unikátního ID. Dále do vrstvy Model patří i třídy určující schéma dat, tzv. POJO[3]. Pomocí těchto tříd jsou převedena (deserializována) data z formátu JSON na objekty jazyka Java. Těmi jsou zejména `Photo`, `Tag`, `PhotosResponse` a další třídy v adresáři `"model"`.

7.1.2 Vrstva View

Vrstvu View v aplikaci tvoří všechny aktivity a fragmenty spolu s jejich rozložením definovaným soubory `layout`. Aktualizace prvků uživatelského rozhraní podle stavu dat je zajištěna funkcionalitou knihovny Data Binding. Díky této knihovně jsem například mohl zjednodušit syntaxi pro přístup k objektům grafických prvků z aktivity. Místo používání metody `findViewById(R.id.view)`, lze přistupovat k těmto objektům přímo zápisem `binding.view`. Tato knihovna také přidává objekty `ObservableField`, které budou popsány v následující kapitole. Dále je využita knihovna Simple Statefull Layout pro globální změny stavu zobrazení. Logika, na základě které je přepínán stav, je ovšem oddělena do vrstvy ViewModel, což je v této architektuře vhodné. Pokud aplikace čeká na data, je ve vrstvě ViewModel nastaven stav na `PROGRESS`. Ve chvíli, kdy jsou připravena data, tento stav je změněn na `CONTENT`. Případně může být aktivován stav `OFFLINE`, když dojde k chybě. Aktivita díky výše zmíněné knihovně Data Binding na tyto změny stavu naslouchá a přepíná zobrazení.

7.1.3 Vrstva ViewModel

Vrstvu ViewModel zastupují dvě třídy a to `MainDataViewModel` a `TagsViewModel`. První zmíněná se stará o logiku hlavních částí aplikace. Aktivita `MainActivity` ji využívá pro získávání dat seznamu obrázků. Tento ViewModel je ale také využíván k zasílání hodnocení těchto obrázků. Druhý zmíněný TagsViewModel je určen pro práci s klíčovými slovy. Toho využívá aktivita `ChooseTagsActivity` pro získávání seznamu klíčových slov i naopak pro odesílání vybraných klíčových slov. Obě třídy této vrstvy využívají k práci s daty třídu `PhotosRepository`. Architektura je znázorněna na obrázku 7.1.

7.2 Uživatelské rozhraní aplikace

Protože aplikace obsahuje menší množství avšak komplexnějších obrazovek (aktivit), bylo nutno věnovat více pozornosti využití jednotlivých komponent.

7.2.1 DetailPhotoActivity

Aktivita obsahující detail obrázku používá zejména větší množství reakcí na doteky. Důvodem jsou využívaná uživatelská gesta. Typickým problémem je konflikt naslouchajících metod na doteky (`OnTouchListener`, `OnClickListener` apod.). Při využití více prvků, které

[2]Shared Preferences je jednoduchý způsob pro trvalé uložení dat v aplikaci. Každý záznam je identifikován svým klíčem a obsahuje jednu hodnotu (key-value pair). Více vysvětluje dokumentace Android Developers, dostupná z: https://developer.android.com/training/data-storage/shared-preferences

[3]POJO (Plain old Java object) je termín pro jednoduchý objekt jazyka Java, který nemá přímou závislost na frameworku a může existovat zcela samostatně.

nějakým způsobem měnily odezvu na doteky, docházelo k chybám, či komplikacím při používání. Celé uživatelské prostředí je uvnitř vertikálně posuvného kontejneru `ScrollView`. Ten obsahuje spodní panel s informacemi a tlačítky, ale také horizontálně přepínatelný kontejner `ViewPager` s fragmenty jednotlivých stránek. Každý fragment zobrazuje obrázek v elementu `TouchImageView` umožňujícím řadu uživatelských gest. Dále celou tuto aktivitu je možno zavřít kromě klasických způsobů (šipka zpět v horním rohu, systémové tlačítko zpět) i potažením nahoru či dolu (toto zajišťuje knihovna `Overscroll Decor`). Pro toto gesto jsem nastavil animaci zmenšení celé aktivity pomocí View Property Animation. Při následném uvolnění prstu je s další přechodovou animací aktivita ukončena. Pro tuto komplexní funkcionalitu jsem musel vyzkoušet mnoho vlastních řešení i knihoven. Výsledné knihovny jsou popsány v kapitole 5. Jedním z problémů byla i poslední zmíněná přechodová animace (Transition), která je blíže popsaná v kapitole 7.4.

7.2.2 MainActivity

Jiné problémy bylo nutno řešit v hlavní aktivitě, tvořené zejména kontejnerem `RecyclerView` s obrázky. Do tohoto kontejneru jsou obrázky postupně doplňovány podle konceptu stránkování (pagination). To je implementováno naslouchající třídou `OnScrollListener`, která při přiblížení ke konci listu požádá `MainDataViewModel` o následující stránku dat. Ve chvíli, kdy vrstva ViewModel obdrží nová data, je provedena aktualizace kontejneru `RecyclerView`.

Tato aktivita je využívána pro zobrazování všech typů seznamů obrázků. Jednotlivé její fragmenty slouží pro zobrazení doporučených, nejnovějších a oblíbených obrázků. Současně je ale využívána i pro výpis obrázků vyhledaných pro určené klíčové slovo.

Nejdříve jsem využíval knihovnu `FlexBox`. Tato knihovna umožnila k `RecyclerView` přidat `FlexBoxLayoutManager` a nastavit několik vlastností zobrazení. Hlavní výhodou tohoto způsobu bylo, že snadno umožňoval přizpůsobování velikosti obrázků podle jejich rozlišení a poměrů stran. Bohužel se objevilo poblikávání a jiné chyby zřejmě související s větším množstvím obrázků, které jsou navíc plynule načítány za posouvání. Proto, po domluvě s vedoucím práce, jsem se rozhodl implementovat vlastní řešení a závislost na knihovně `FlexBox` zrušit. Využil jsem k tomu základní `GridLayoutManager`, který je součástí knihovny `Android RecyclerView Support`. Tento layout umožňuje zobrazovat položky jako dlaždice s předem daným počtem sloupců. Navíc je možné u každé položky nastavit tzv. “span”, tedy propojení s určeným počtem následujících položek. Díky této možnosti jsem vytvořil systém přidělování hodnoty “span” všem obrázkům podle jejich poměru stran. K tomu dochází ihned po získání nových dat uvnitř `PhotosRepository`. Následně se tato hodnota použije při nastavování velikosti každého obrázku v listu. Využívám pro to 12 sloupců, protože jde o ideálně dělitelné číslo. Je možno dělit beze zbytku čísly 2, 3, 4 nebo 6. Obrázky na jednom řádku tak můžou mít oba “span” 6 a tím zabírat právě polovinu prostoru řádku. Nebo s odlišnými hodnotami “span” může být jeden obrázek širší než druhý. Případně je možné zobrazit na jednom řádku například 3 obrázky.

7.2.3 SplashActivity

Během otevírání a načítání celé aplikace se běžně nezobrazuje žádný obsah. To lze vyřešit aktivitou, která nemá vlastní layout (soubor s rozložením grafických prvků). Místo toho je její vzhled nadefinován v souboru `"styles.xml"`. `SplashActivity` je nastavena jako tato spouštěcí aktivita aplikace. Obsahuje pouze logo aplikace a při načtení kódu jazyka Java provede přepnutí na jinou aktivitu podle hodnoty v Shared Preferences. Při prvním spuštění je otevřena `IntroActivity`, při dalších už `MainActivity`.

7.2.4 IntroActivity

Tato úvodní aktivita představuje aplikaci Impres na třech stránkách pomocí prvku `ViewPager`. Kromě základní funkce také dokáže plynule prolínat barvu pozadí. K tomu jsem využil třídu `ArgbEvaluator`, což je velmi jednoduchý způsob prolínání barev využívaný i třídou `ObjectAnimator`. Ilustrace v této aktivitě jsem vytvořil s využitím volně dostupných vektorových ikon.

7.2.5 ChooseTagsActivity

Tato aktivita se spouští pouze poprvé a obsahuje kontejner `RecyclerView` s výpisem klíčových slov. Pro práci s daty využívá třídu `TagsViewModel`. Položky s klíčovými slovy mají zaoblené rohy a nastaveny mezery pomocí tříd z kapitoly 5.10. Pozadí položek tvoří obrázky načítané knihovnou Glide.

7.3 Klient API

Velmi důležitou částí služby je komunikace se serverem skrze REST API. Data jsou přenášena ve formátu JSON a parametry metodou GET. Struktura těchto dat je blíže popsána v kapitole 6.2.

Pro sestavení těchto dotazů, odeslání a následný příjem odpovědi jsem využil knihovnu Retrofit 5.2 spolu s OkHttp. Pro HTTP klienta (OkHttp) jsem nastavil `Interceptor` ovlivňující tvorbu paměti cache, tedy tzv. kešování. Při aktivním internetovém spojení je odpověď kešována po dobu 60 sekund, ovšem ve stavu off-line je to až jeden týden. Retrofit zjednodušuje tvorbu dotazů formou rozhraní jazyka Java. Toto rozhraní deklaruje metody využívané pro konkrétní dotazy. Jednou z metod je `getPhotos()`. Ta se využívá pro získání výpisu obrázků. Deklaraci této metody ukazuje výpis kódu 7.1.

```
21    @GET("api-provider/getPhotos.php")
22    Observable<PhotosResponse> getPhotos(
23        @Query("userId") int userId,
24        @Query("source") String source,
25        @Query("sort") String sortBy,
26        @Query("p") int page);
```

Výpis 7.1: Metoda `getPhotos()` s definovanou koncovou adresou serveru (endpoint) anotací na řádku 21. Metodou GET budou předány čtyři parametry. Odpověď na tento dotaz bude deserializována podle třídy `PhotosResponse`.

Odpověď na dotaz je přijata ve formátu JSON a deserializována podle jedné ze tříd modelu. Serializaci a deserializaci provádí knihovna Gson. V ukázce 7.1 se využívá pro odpověď třída `PhotosResponse`, proto tento případ blíže popíši. Tato třída obsahuje tři pole typu `Integer`, obsahující celkový počet vyhovujících záznamů v databázi, počet výsledků v této odpovědi a číslo aktuální strany. Aplikace totiž provádí načítání postupně a využívá k tomu koncept stránkování (pagination). Když si aplikace vyžádá další data, tak je odeslán dotaz s číslem následující strany. Ukázka třídy `PhotosResponse` 7.2 obsahuje tato

celočíselná pole spolu se seznamem fotografií `List<Photo> photos`. Konkrétní příklad dat ve formátu JSON zobrazuje kód 7.3.

```
16 public class PhotosResponse extends BaseObservable implements Serializable {
17
18     @SerializedName("totalCount")
19     @Expose
20     private Integer totalCount; // total count of photos for this request
21     @SerializedName("resultCount")
22     @Expose
23     private Integer resultCount; // count of photos in this response
24     @SerializedName("page")
25     @Expose
26     private Integer page; // actual page number
27     @SerializedName("photos")
28     @Expose
29     private List<Photo> photos = null;
30     ...
```

Výpis 7.2: Ukázka ze třídy `PhotosResponse`. Tato třída je využita pro deserializaci odpovědi serveru ve formátu JSON. Proto implementuje rozhraní `Serializable`. Rozšíření třídy `BaseObservable` umožňuje sledování změn objektu. Podle anotace `SerializedName` je převedeno pole z JSON do konkrétního pole této třídy.

```
1  {
2      "totalCount": 4708,
3      "resultCount": 30,
4      "page": 1,
5      "photos": [
6          {
7              "id":"40820535375",
8              "source":"Flickr.com",
9              "owner":"73082187@N07",
10             "owner_name":"KHR Images",
11             "title":"Cuckoo",
12             "dateupload":"1524760353",
13             "views":"0",
14             "tags":"cuckoo cuculuscanorus wild bird mature male migrant
                   surrey rain aprilshower wildlife nature nikon",
15             "url_o":"",
16             "url_l":"https:\/\/farm1.staticflickr.com\/832\/40820535375
                   _33172d5164_b.jpg",
17             "height_o":"611",
18             "width_o":"1024",
```

```
            "url_m":"https:\/\/farm1.staticflickr.com\/832\/40820535375
                _33172d5164.jpg",
            "height_m":"298",
            "width_m":"500",
            "time_created":"2018-04-27 11:37:54",
            "relationship_sum":"6720"
        },
        {
            ...
        }
    ]
}
```

Výpis 7.3: Ukázka konkrétních dat ve formátu JSON. Tato data jsou poskytována serverem při dotazu na koncový bod (endpoint) "api-provider/getPhotos.php". Obsah je zkrácen.

Kromě knihoven Retrofit a OkHttp jsem využil také RxJava 5.3 a adaptér pro Retrofit. Díky tomu je možné provádět dotaz v jiném vlákně na pozadí a následně odpověď ze serveru obdržet v námi definovaném vlákně. Pro zpracování odpovědi je využita třída `Observer` s metodami `onNext()` a `onError()` pro zpracování odpovědi nebo chyby a předání vrstvě ViewModel.

7.4 Animace

V aplikaci jsem využil řadu animací. Kde to bylo možné, použil jsem modernější přístup se třídou `ViewPropertyAnimator`. Některé objekty v aplikaci obsahují specifické animace, které jsem definoval v XML. Tyto animace se nacházejí v adresáři `"res/anim"` a pomocí nich se točí ikona v aktivitě `ChooseTagsActivity` nebo animuje ikona srdce v `DetailPhotoActivity`.

Pro animovaný přechod do nové aktivity jsem využil Android Transition framework , který je součástí Android API od verze 21 (Android 5.0). Kromě běžné vstupní (`"windowEnterTransition"`) a výstupní (`"windowExitTransition"`) animace prvků aktivity jsem využil také Shared Element Transition. Toto slouží k plynulému přenesení elementu z původní aktivity do nově otevřené. Ve skutečnosti nedochází k přenesení elementu `View` z jedné aktivity do druhé. Ve stromě každé aktivity jsou oba elementy samostatně. Tento typ animace pouze získá z cílového elementu jeho pozici a rozměry pro následné provedení přechodové animace. Pro správnou funkčnost jsem nastavil u všech obrázků unikátní hodnotu `"transitionName"`, která musí být nastavena na totožnou hodnotu i v cílové aktivitě, do které bude přechod proveden. Následně jsem s využitím `SharedElementCallback` nastavil konkrétní sdílený prvky při kliknutí. Toto umožnilo animovat přechod i zpětně.

Základní implementace ovšem nestačila, bylo potřeba zajistit chování i u složitějších případů [12]. Při změně obrázku v detailu (což je možné provést posunem do boku kontejneru `ViewPager`) a následném návratu do hlavní aktivity jsem chtěl zajistit, aby se zpětně animoval obrázek aktuální a ne původní. Pro tuto funkcionalitu jsem doplnil několik potřebných částí. Po změně obrázku v detailu je automaticky novému obrázku přiřazeno `"transitionName"`. Při zavření aktivity dojde ke zpětnému volání (callback), které dostane informaci o aktuální poloze obrázku v listu a do seznamu `sharedElements` je dopl-

něn správný element grafického rozhraní (View) se správným jménem `"transitionName"`. Tímto je možno provádět přechodové animace mezi listem obrázků a detailem obrázku, i když detail umožňuje přepínat obrázky.

U Android transition framework se ale objevily i nedostatky a chyby. Ne vždy se jedná o spolehlivou technologii, což potvrzuje například diskuze vývojářů[4]. V aplikaci Impres je možno zaznamenat výjimečně chybu této animace. Při této chybě dojde k chvilkové chybné animaci, po které následuje správná animace elementu. Také je možno zaznamenat zamrznutí obrázku před provedením animace. V současné době bohužel neexistuje vhodná všestranná náhrada, a proto jsou pravděpodobně jen dvě možnosti. Nevyužívat tyto přechody (což evidentně zvolili aplikace z kapitoly 3) nebo vytvořit vlastní řešení, které by bylo ovšem velmi komplikované.

7.5 Další pomocné třídy

Vytvořil jsem pomocnou nástrojovou třídu (utility class) `ImpresUtil`. Původně se jmenovala `AppConfig` a jednalo se o návrhový vzor jedináček (singleton). S tímto návrhovým vzorem je ale na platformě Android několik problémů, a proto se příliš nedoporučuje. Setkal jsem se v aplikaci s problémy, jako nepředvídatelnou ztrátou stavu způsobenou systémem. Podobné problémy popisuje i Michael Spitsin [14]. Proto jsem se rozhodl nahradit tuto třídu bezstavovou (stateless) nástrojovou třídou, která obsahuje pouze statické metody. Pokud daná metoda vyžaduje kontext (třída `Context`), je předáván parametrem.

[4] https://www.reddit.com/r/androiddev/comments/73x84q/im_done_with_the_android_transitions_api/

Kapitola 8

Implementace serverové aplikace

Kromě klientské mobilní aplikace se služba Impres skládá i ze serverové aplikace, vytvořené v jazyce PHP. Ta se skládá z několika modulů v oddělených adresářích. Data Saver (`"data-saver/"`) se stará o získávání a ukládání dat. Api Provider (`"api-provider/"`) naopak data z databáze poskytuje přes veřejné REST API klientské aplikaci.

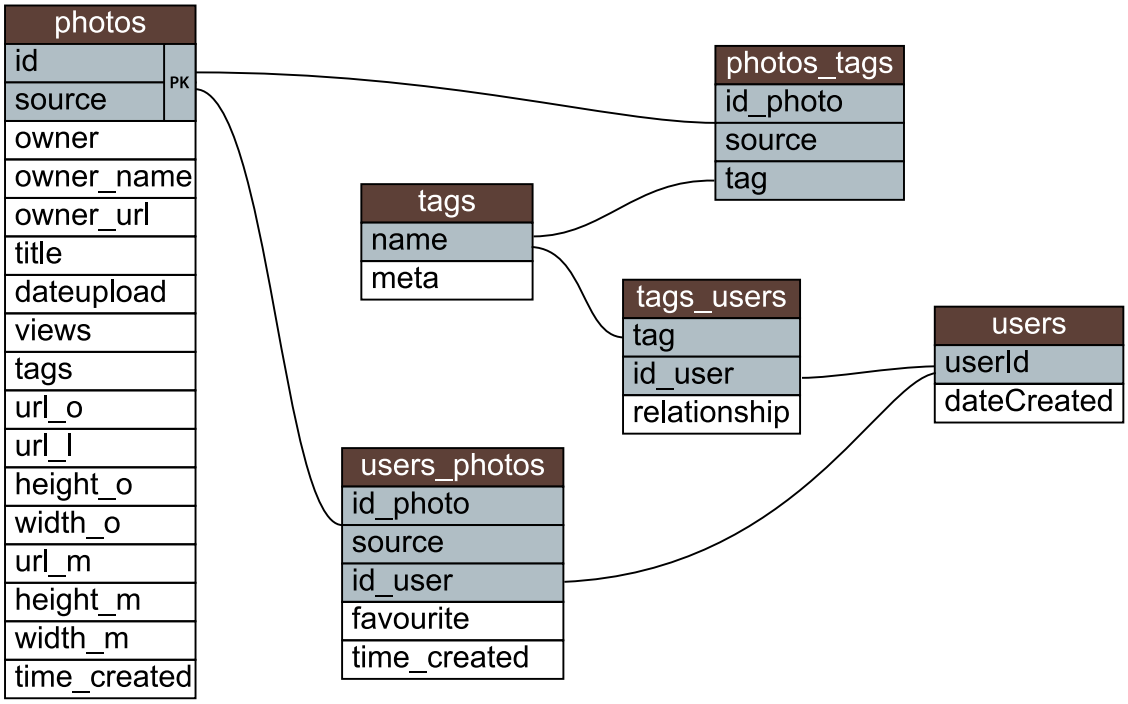
8.1 Databáze

Pro uchování dat na serveru využívám relační databázi MySQL. Jazyk tohoto systému vychází z deklarativního jazyka SQL (Structured Query Language). Výhodou MySQL je uplatnění známého relačního databázového modelu a současně skutečnost, že se jedná o open-source dostupný zdarma. Dále je možné zvolit tzv. motor (engine). Mezi nejčastěji využívané motory MySQL patří MyISAM a InnoDB. Druhý jmenovaný sice podporuje transakce a je vhodnější pro práci s většími objemy dat, ale zase nedosahuje takové rychlosti jako MyISAM. MyISAM je tedy rychlejší a jednodušší variantou optimalizovanou pro větší množství dotazů `SELECT`. Zvolil jsem právě MyISAM, protože služba Impres neobsahuje příliš složité databázové schéma ani komplikovanější operace vyžadující transakce. Důležitá je rychlost zpracovávání většího množství jednodušších operací typu `SELECT`.

Databáze uchovává zejména všechny obrázky v tabulce `photos`. Při ukládání nových obrázků jsou vkládány jejich klíčová slova do tabulky `tags`. Uživatelské identifikátory jsou potom v tabulce `users`. Mezi záznamy těchto tabulek vznikají vztahy, pro které existují v databázi další dvě tabulky. `Photos_tags` určuje přiřazení klíčových slov k obrázkům. Tabulka `tags_users` obsahuje všechny vztahy mezi uživatelem a klíčovým slovem, podle kterých je prováděno doporučování obsahu. Nakonec `users_photos` se využívá pro uložení oblíbených obrázků každého uživatele. Tyto tabulky a jejich vztahy znázorňuje obrázek 8.1.

8.2 Získávání a zpracování dat z REST API více služeb

Protože každé API má jiný formát nebo i způsob zabezpečení, je nutno vytvořit systém, který dokáže být s těmito API kompatibilní. Nenašel jsem řešení, které by se přímo tomuto věnovalo, proto jsem vytvořil vlastní způsob. Vždy je nutno prostudovat dané API a připravit pro něj celý postup od tvorby dotazu až po zpracování různých odpovědí. To celou záležitost samozřejmě komplikuje. Cílem tedy bylo co nejvíce kroků zjednodušit a zautomatizovat. Výstupem tohoto zpracování musí být jednotná struktura dat odpovídající schématu databáze.



Obrázek 8.1: Schéma serverové databáze. Vztahy mezi tabulkami ve skutečnosti použitá verze MySQL s motorem (engine) MyISAM nepodporuje. Zde jsou vztahy naznačeny pro lepší pochopení použití tabulek.

Prvním krokem je třída `ApiClient`, která obsahuje samostatné metody pro každou službu. V této metodě je sestaven a proveden dotaz na API konkrétní služby v definovaném tvaru, jaký dané API vyžaduje. K tomu využívám knihovnu GuzzleHttp. Výsledkem této metody je odpověď serveru ve formátu JSON.

Zpracování těchto strukturovaných dat provede třída `DataSavingUtilities`. Vytvořil jsem metodu `findRecursiveArrayNodeById()`, která provádí rekurzivní vyhledávání v komplikovaných strukturovaných datech a dokáže přiřazovat nalezené hodnoty do výsledné struktury (odpovídající DB). Vstupem této metody je speciální vícerozměrné pole odpovídající struktuře dat daného API. Pomocí tohoto pole je definováno, které hodnoty vstupních dat budou využity, a do kterých polí databáze budou předány. Na ukázce kódu 8.1 je vidět toto pole pro konkrétní zpracování dat ze služby Unsplash.com. Pro některé služby je navíc vhodné provést dodatečné zpracování. U služby Unsplash.com například provádím doplnění klíčových slov z popisu (pole `"description"`), protože obsahuje vhodná klíčová slova, která jinak přes API nelze získat.

Spouštění celého tohoto procesu je řízeno ze souboru `data-saver/run.php`. Tento skript se postará o zavolání potřebných metod pro získání a následně i zpracování dat. Skript je možno spouštět ručně nebo automatizovaně pomocí technologie CRON[1]. Díky tomu může být zajištěna snadná periodická aktualizace dat.

```
1 $SOURCE = "Unsplash.com";
```

[1]Cron je softwarový démon, který automatizovaně spouští v určitý čas nějaký příkaz resp. proces. Detailnější vysvětlení je dostupné z URL: https://cs.wikipedia.org/wiki/Cron

```
$values = array(
    'id' => 'id',
    'user' => array('username' => 'owner',
        'name' => 'owner_name'),
    'description' => 'title',
    'created_at' => 'dateupload',
    'categories' => 'tags',
    'urls' => array('small' => 'url_m',
        'full' => 'url_l',
        'raw' => 'url_o'),
    'height' => 'height_o',
    'width' => 'width_o'
);
```

Výpis 8.1: Pole pro převedení zdrojové struktury dat získaných z Unsplash.com API do podoby požadované databází. Klíče tohoto pole odpovídají struktuře a názvům vstupních dat formátu JSON. Hodnoty tohoto pole jsou požadované názvy pro uložení do databáze. Například pro uložení jména autora do databázového pole `"owner_name"` bude získána hodnota `"name"` ze zanořeného pole `"user"`.

8.3 Poskytování dat přes REST API

Oproti získávání a ukládání nových dat je tato část o něco jednodušší. Při dotazu na konkrétní koncový bod je spuštěn PHP skript, který nejdříve zkontroluje přítomnost vstupních parametrů. Následně je podle těchto parametrů rozhodnuto jaký dotaz (query) do databáze bude proveden. Pro práci s databází jsem vytvořil třídu `DatabaseHelper`. Například skript `"getPhotos.php"` vyžaduje parametry `"p"` (číslo strany), `"tag"` (klíčové slovo pro filtrování) a `"sort"` (typ řazení). Tyto hodnoty jsou ošetřeny buďto metodou `mysqli::escape_string()` nebo převedením na celé číslo metodou `intval()`.

Po provedení dotazu jsou získaná data z databáze převedena na formát JSON pomocí `json_encode($db->result)`. Tento výstup je vypsán a v kombinaci s hlavičkou `header('Content-Type: application/json');` je připraven pro automatické čtení klientskou aplikací.

8.4 Systém personalizace obsahu

Personalizace obsahu je implementována na serveru pomocí dotazů do databáze MySQL. Při obdržení zprávy na koncový bod `getPhotos.php` s hodnotou parametru `"sort=personalized"` je spuštěn tento dotaz. Dále je pro správnou funkčnost doporučování obsahu využíván parametr `"userId"`. Pro jednoduchost a rychlost zpracování, je celý algoritmus (popsaný v kapitole 6.2) implementován v jednom složeném dotazu. Tento dotaz ukazuje výpis kódu 8.2.

```
SELECT photos.*, ROUND(AVG(tu.relationship)) relationship_sum
    FROM (
```

```
3           SELECT *
4           FROM photos
5           ORDER BY photos.time_created DESC
6           LIMIT N
7           ) photos
8       LEFT JOIN photos_tags pt ON pt.id_photo = photos.id
9       LEFT JOIN (
10          SELECT *
11          FROM tags_users
12          WHERE id_user=$idUser
13          ) tu ON tu.tag = pt.tag
14
15      GROUP BY id
16      ORDER BY relationship_sum DESC;
```

Výpis 8.2: MySQL dotaz pro personalizovaný seznam obrázků. Hodnota `"N"` je v současnosti nastavena na 500. Vyžadovaná proměnná je ID uživatele.

Kromě tohoto dotazu je potřeba také přijímat data s hodnocením fotografií z klientských aplikací. O tuto záležitost se stará skript `putRating.php`, který obdrží ID fotografie a číselné hodnocení, kterým upraví vztah všech klíčových slov obrázku k danému uživateli. Skript `putTagsRating.php` slouží k přijímání hodnocení více klíčových slov. Toto je využíváno u obrazovky mobilní aplikace `ChooseTagsActivity` pro odeslání uživatelem vybraných klíčových slov na server.

Kapitola 9

Závěr

Cílem této bakalářské práce bylo přiblížit problematiku vývoje mobilních aplikací pro operační systém Android, ale také navrhnout a implementovat mobilní aplikaci Impres se serverovou aplikací. Tyto cíle se podařilo naplnit a vytvořit funkční mobilní aplikaci. Ta splňuje vytyčené body z kapitoly 6.1 a byla publikována na oficiálním obchodě Google Play v režimu veřejného beta testování[1]. Bude vhodné pokračovat v testování a odladění pro různé typy zařízení.

Naučil jsem se pracovat s různými formáty veřejných API, a tak lépe pochopil principy s tím spojené. Zjistil jsem, že některá API pro tuto službu nejsou vhodná, proto jsem je ve výsledném řešení nevyužil. Pro získávání dat z více serverů jsem navrhl vlastní řešení v jazyce PHP, které funguje pro vybraná API a bude možné doplňovat další. V současnosti jsou využívány data ze služeb Flickr.com, Unsplash.com, DeviantArt.com a Artsy.com. V budoucnu bych rád přidal více zdrojů klasického umění.

Díky studiu problematiky vývoje aplikací pro operační systém Android jsem si značně prohloubil dosavadní zkušenosti. Během vývoje aplikace Impres jsem se mnohokrát setkal se zásadními problémy nekompatibility knihoven nebo návrhových vzorů. Využil jsem řadu zdrojů pro studium jednotlivých částí implementace, a přesto většinu kódu opakovaně přepisoval a měnil. Celý vývoj tedy probíhal v zamýšlených ale i neplánovaných iteracích. Zjistil jsem, že je velmi zásadní sledovat aktuální doporučené postupy společnosti Google, která je prezentuje různými kanály, jako jsou oficiální příručky, články zaměstnanců společnosti, vývojářské skupiny na Google+ nebo záznamy z konference Google I/O. Většina zdrojů na internetu neobsahuje již zcela aktuální pohled na vývoj pro tento operační systém, a proto může být vyhledávání informací problematické.

Službu Impres je možno dále rozšiřovat nad rámec plánovaný pro tuto práci. Tato rozšíření by se mohla týkat pokročilejšího systému doporučování obsahu. To bude možné zejména u většího množství uživatelů, kdy půjde zapojit i jiné typy těchto systémů. Dále by služba Impres mohla být rozšířena o webovou verzi. Vhodně navržená komunikace mobilní a serverové aplikace umožňuje vyvíjet tyto aplikace samostatně, případně doplňovat další klientské aplikace pro další platformy.

[1] https://play.google.com/store/apps/details?id=cz.davidkocnar.impres

Literatura

[1] App Stores: Number of apps available in leading app stores as of March 2017. *Statista*. [Online; navštíveno 11.05.2018].
URL https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

[2] Google. Android Architecture Components. *Android Developers*. [Online; navštíveno 5.05.2018].
URL https://developer.android.com/topic/libraries/architecture/

[3] Google. Guide to App Architecture. *Android Developers*. [Online; navštíveno 11.05.2018].
URL https://developer.android.com/jetpack/docs/guide

[4] Google. Platform Architecture. *Android Developers*. [Online; navštíveno 13.05.2018].
URL https://developer.android.com/guide/platform/

[5] REST Principles and Architectural Constraints. *RestfulApi*. [Online; navštíveno 14.05.2018].
URL https://restfulapi.net/rest-architectural-constraints/

[6] C. Aggarwal, C.: *Recommender Systems*. Springer International Publishing, 2016, ISBN 978-3-319-29659-3.

[7] Deering, S.: Do you know what a REST API is?. *SitePoint*. [Online; navštíveno 14.05.2018].
URL https://www.sitepoint.com/developers-rest-api/

[8] Joshi, H.: Be Reactive: Develop your next app with RxJava. *AndroidPub*.
URL https://android.jlelse.eu/be-reactive-develop-your-next-app-with-rxjava-4f00bfde0020

[9] Klimov, I.: Android Animation Types Explained. *MyHexaville*. [Online; navštíveno 13.05.2018].
URL
http://myhexaville.com/2017/01/11/android-animation-types-explained/

[10] Kulka, J.: *Psychologie umění*. Státní pedagogické nakladatelství, n.p., 1991, ISBN 80-04-23694-4.

[11] Kumar, V.: Android Architecture Patterns : MV(C | P | VM). *Medium.com*, Prosinec 2017.

[12] Mion, A.: "Dynamic" Shared Element Transition: How to hold a common gallery flow. *AndroidPub*. [Online; navštíveno 11.05.2018].
URL https://android.jlelse.eu/dynamic-shared-element-transition-23428f62a2af

[13] Sharma, A.: Why to choose MVVM over MVP – Android Architecture. *AndroidPub*. [Online; navštíveno 13.05.2018].
URL https://android.jlelse.eu/why-to-choose-mvvm-over-mvp-android-architecture-33c0f2de5516

[14] Spitsin, M.: Singletons in Android. *Medium.com*, Červenec 2016.

[15] Wouts, F.: Choosing the right technology for your mobile app. *Medium.com*, Leden 2018.

[16] Wurmanová, N.: Odkud přichází inspirace? *Psychologie.cz*, Říjen 2011.

Příloha A

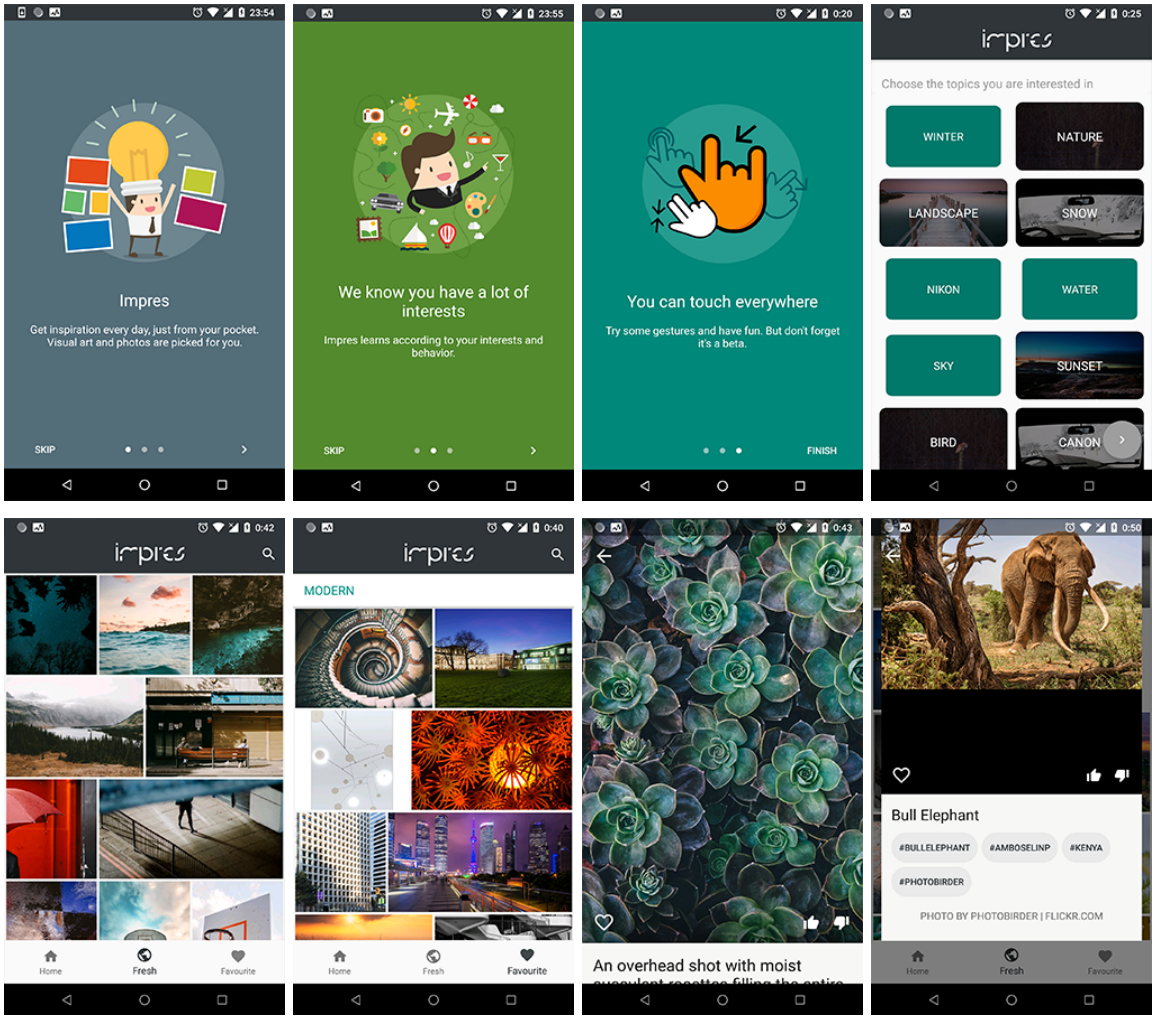
Obsah přiloženého paměťového média

Přiložené CD obsahuje tyto adresáře:

- klient-apk – instalační soubor mobilní aplikace
- klient – všechny zdrojové soubory mobilní aplikace
- server – všechny zdrojové soubory serverové aplikace
- obrazova-priloha – plakát a snímky obrazovek
- video – demonstrační video mobilní aplikace
- pdf – text práce ve formátu PDF
- latex – zdrojové kódy textu práce
- manual – pokyny k obsahu CD a instalaci

Příloha B

Obrazovky mobilní aplikace



Obrázek B.1: Ukázka obrazovek mobilní aplikace ve snížené kvalitě. První řada zleva: tři kroky úvodního představení aplikace, obrazovka pro výběr oblíbených témat. Druhá řada: nejnovější obrázky (fresh), obrázky pro klíčové slovo “modern”, detail obrázku, detail obrázku zavíraný gestem tažení zespodu.

Příloha C

Plakát

Obrázek C.1: Propagační plakát mobilní aplikace Impres ve snížené kvalitě.