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 EVIDENCI A ROZDĚLOVÁNÍ SPROPITNÉHO V RESTAURACI

MOBILE APP FOR MANAGING TIPS IN A RESTAURANT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MARTIN KOCOUR**
AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. ADAM HEROUT, Ph.D.**
SUPERVISOR

BRNO 2017

Zadání bakalářské práce/20035/2016/xkocou08

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Kocour Martin**
Obor: Informační technologie
Téma: **Mobilní aplikace pro evidenci a rozdělování spropitného v restauraci**
Mobile App for Managing Tips in a Restaurant
Kategorie: Uživatelská rozhraní

Pokyny:

1. Prostudujte problematiku návrhu uživatelských rozhraní a tvorby aplikací pro Android.
2. Prostudujte a analyzujte problematiku výběru spropitného v restauraci.
3. Navrhněte principy uživatelského rozhraní pro správu a rozdělování spropitného.
4. Implementujte minimalistickou použitelnou verzi řešené aplikace.
5. Testujte aplikaci na uživatelích z restaurační praxe.
6. Analyzujte zpětnou vazbu uživatelů a navrhněte+realizujte iterativní úpravy řešené aplikace.
7. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- dle pokynů vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3,
- značné rozpracování bodu 4,
- rozpracování bodů 5 a 6.

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 2016
Datum odevzdání: 17. května 2017

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

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Táto bakalárska práca sa zaoberá problematikou evidencie sprepitného. Výsledkom je mobilná aplikácia pre tablety, ktorá umožňuje jednoducho zaznamenávať získané finančné odmeny a následne ich transparentne deliť medzi zamestnancov. Táto práca v úvode pojednáva o problematike tvorby mobilných aplikácií pre operačný systém Android. Ďalej sa čitateľ dozvie, akým spôsobom bola aplikácia navrhnutá a implementovaná. V závere sú zhrnuté výsledky z testovania a budúce smerovanie vývoja aplikácie.

Abstract

This bachelor thesis deals with an issue of tip evidence. The result is an Android application for tablets, which provides easy tip recording and its transparent division among employees. In the thesis are described problems with developing application for Android operating system. Reader will read about the process of designing and implementing. The paper is supplemented with results of application testing and its future in way of developing.

Kľúčové slová

Android, aplikácia, sprepitné, reštaurácia, bar, kaviareň, čašník, evidencia, EET

Keywords

Android, app, tips, restaurant, pub, coffee shop, waiter, evidence, EET

Citácia

KOCOUR, Martin. *Mobilní aplikace pro evidenci a rozdělování spropitného v restauraci.* Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Herout Adam.

Mobilní aplikace pro evidenci a rozdělování spropitného v restauraci

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána prof. Ing. Adama Herouta, Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Martin Kocour
16. mája 2017

Poďakovanie

Rád by som sa poďakoval svojmu vedúcemu práce prof. Ing. Adamovi Heroutovi, Ph.D. za odborné vedenie tejto bakalárskej práce, cenné rady a konštruktívnu kritiku, ktorá viedla k úspešnému vypracovaniu práce.

Obsah

Kapitola 1

Úvod

V Českej republike bol nedávno schválený zákon o elektronickej evidencii tržieb (zk. EET) po vzoroch z Chorvátska a Slovinska. Tento zákon je v platnosti od 1. Decembra 2016 a ukladá podnikateľom povinnosť evidovať všetky svoje tržby a odosielať ich na finančnú správu. Hlavným zámerom štátu je zefektívniť výber daní a zamedziť tak rastu šedej ekonomiky, ktorá sa medzi podnikateľmi rozmohla.

V súvislosti s EET vznikol nový priestor pre predajcov pokladných systémov. Po novom musí mať každý subjekt, ktorého sa zákon týka, pokladňu s podporou evidencie tržieb. Z trhu tak postupne miznú jednoúčelové tlačítkové pokladne a nahrádzajú ich nové viacúčelové. Aktuálny trend medzi predajcami je taký, že svoje riešenie vyvíjajú prevažne na mobilných zariadeniach s operačným systémom Android. Vďaka tomu sa na českom trhu otvára nový priestor pre tzv. B2B[1] mobilné aplikácie.

Na tento fakt sa snaží nadviazať aj táto bakalárska práca. Úlohou môjho projektu je vytvoriť aplikáciu pre systém Android, ktorá bude evidovať, teda zaznamenávať získané sprepitné od spokojných zákazníkov a následne ho spravodlivo deliť medzi ľudí, ktorí sa na získanej odmene spoločne podieľali. Hlavnou myšlienkou je vytvoriť transparentný systém, ktorý umožní svojím používateľom mať prehľad o tom, aké množstvo sprepitného sa vyzbieralo. Zmyslom je tak predísť frustrácii zamestnancov, ktorí majú obavy z toho, že ich zamestnávateľ oberá o časť odmeny, ktorú získali. Motiváciou pre zamestnávateľov by mal byť tiež fakt, že akákoľvek forma prijatej finančnej odmeny podlieha tiež daneniu nezávisle na tom, kto ju získal. V záujme zamestnávateľa by teda mal byť kladený dôraz takúto odmenu riadne zaevidovať.

Táto práca v prvej časti pojednáva o problematike vývoja mobilných aplikácii pre OS Android, kapitola 2. Je tu spomenutá nie len architektúra operačného systému ale aj časté problémy, s ktorými sa vývojár stretne. Poznatky z tejto časti sú ďalej využité pri návrhu aplikácie a jej implementácii. Návrh aplikácie sa rozoberá v kapitole 3, kde je bližšie rozpísaný databázový model a používatelské rozhranie spolu s analýzou existujúcich riešení. Ďalšia kapitola 4 je venovaná implementácii, kde sa čitateľ dozvie podrobnejšie, ako sú jednotlivé časti aplikácie realizované. V závere, v kapitole 7, je analyzovaný celkový výsledok práce.

[1] B2B – business-to-business (možno voľne preložiť ako obchod pre iný obchod), skratka označujúca vzťah medzi dvoma a viac obchodníkmi, viď [5].

Kapitola 2

Vývoj aplikácií pre operačný systém Android

S Androidom sa dnes môžeme stretnúť nielen v mobilných telefónoch alebo v tabletoch, ale už aj v autách, v televíziách a v hodinkách. Tento operačný systém sa stal populárny najmä vďaka otvoreným zdrojovým súborom. Výrobcovia si ho tak môžu upraviť podľa svojich potrieb. Tento projekt vyvíja spoločnosť Google Inc., ktorá taktiež založila konzorcium Open Handset Alliance na správu a vydávanie nových štandardov pre mobilné zariadenia.

Táto kapitola popisuje základné princípy tvorby mobilných aplikácií. Čitateľ sa dozvie o architektúre systému (kap. 2.1), jeho základných komponentoch (kap. 2.2) a o novom vizuálnom jazyku Material Design (kap. 2.4). V závere tejto kapitoly opisujem časté problémy pri vývoji a ich riešenia (kap. 2.6). Všetky tieto teoretické základy využívam v návrhu a implementácii tejto práce. Informácie v tejto kapitole som prevzal z kníh *Průvodce programovaním mobilných aplikací* [6], *The Busy Coder's Guide to Android Development* [7] a z oficiálnej príručky projektu Android [2].

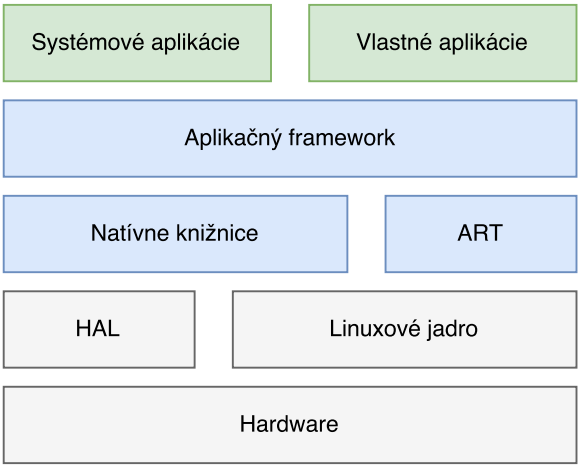
2.1 Architektúra operačného systému

Operačný systém Android je založený na linuxovom jadre. Celá architektúra je rozčlenená do niekoľkých vrstiev, ako je to vidieť na obrázku 2.1:

- **Systémové aplikácie** – Android vo svojom základe ponúka množinu systémových aplikácií. Táto množina sa môže líšiť podľa konkrétneho výrobcu zariadenia, ktorý si ju môže ľubovoľne upraviť. Ide o aplikácie na správu emailov, správ, kontaktov a iné.
- **Aplikačný framework** – súbor nástrojov pre vývoj Android aplikácií. Je napísaný v jazyku Java a zapúzdruje triedy na prácu s notifikáciami, so systémom GPS a mnoho ďalších.
- **Natívne knižnice** – základné systémové komponenty, ako napr. OpenGL, sú postavené na natívnom kóde, ktorý vyžaduje knižnice napísané v C/C++. Väčšina funkcionality z týchto knižníc je zaobalená práve v aplikačnom frameworku, ktoré poskytuje svoje triedy na komunikáciu s týmito knižnicami.
- **Android Runtime** (ART) – virtuálny stroj na spracovanie *dex* súborov. Android vyvinul svoj vlastný bajtkódový formát dex, ktorého úlohou je efektívnejšie pracovať s operačnou pamäťou. Zdrojové súbory každej aplikácie sa skompilujú napr. pomocou

nástroja Jack[1] na dex-ový bajtkód a zabalia sa spolu s ďalšími súbormi do jedného aplikačného archívu. Predchodcom ART bol virtuálny stroj Dalvik, viď [1].

- **Hardware Abstraction Layer** (HAL) – poskytuje štandardné rozhranie na komunikáciu s hardvérovými komponentmi ako je zvuk, kamera, Bluetooth a ďalšie. Výrobcovia mobilných zariadení môžu toto rozhranie rozšíriť so svojou implementáciou ovládačov. To im umožní jednotne komunikovať so svojimi špecifickými perifériami cez Aplikačný framework.
- **Linuxové jadro** – mierne upravené linuxové jadro (kernel) pre potreby mobilných zariadení. Rozdiel je hlavne v správe pamäti a správe procesov.



Obr. 2.1: Popis architektúry platformy Android

2.2 Základné komponenty

Aktivita

Aktivita je základný stavebný blok pri budovaní používateľského rozhrania. Môžeme si ju predstaviť ako entitu systému Android analogickú k oknu v klasickom desktopovom svete alebo ako stránku vo webových prehliadačoch. Bežne aktivita zaberá takmer celú obrazovku, ale od verzie 7.0 je možné obrazovku rozdeliť na dve časti a zobraziť tak naraz dve aktivity rôznych aplikácií (tzv. „split screen mode“), viď [7]. Aplikácie sa bežne skladajú aspoň z jednej aktivity.

Životný cyklus aktivity

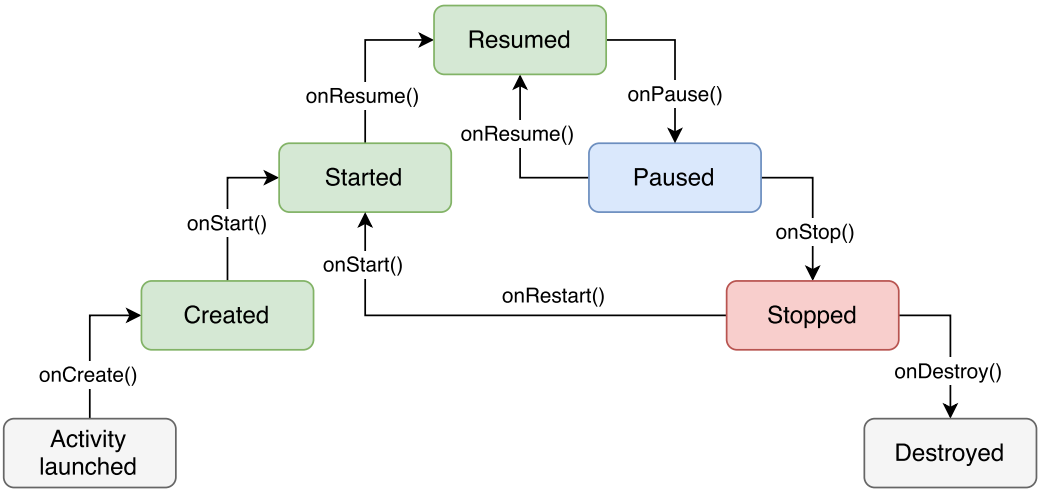
Každá aktivita má svoj životný cyklus, ktorý je úzko spätý s používaním mobilného zariadenia. Celý priebeh cyklu zachytáva obrázok 2.2. Aktivita sa môže dostať do nasledujúcich stavov:

- **Created** – po spustení aktivity sa zavolá metóda `onCreate()`, ktorej úlohou je najmä vytvoriť používateľské rozhranie a načítať zobrazované dáta. Táto metóda sa volá iba raz za celý cyklus.

[1] Jack toolchain – https://source.android.com/source/jack

- **Started** – metóda `onStart()` je volaná nie len po vytvorení aktivity, ale aj keď sa aktivita vráti opäť do popredia. To je hlavný rozdiel medzi `onCreate()` a `onStart()`.
- **Running** – akonáhle je aktivita vytvorená a inicializovaná tak pomocou metódy `onResume()` prechádza do stavu *Running*[2], v ktorom už používateľ môže interagovať s aktivitou. V tomto stave aktivita ostáva až do chvíle, kým používateľ buď neotočí obrazovku alebo neprejde do inej aktivity.
- **Paused** – doplnkom k metóde `onResume()` je `onPause()`. Do stavu *Paused* sa aktivita dostane v prípade, že:
  - Aktivita prechádza do pozadia.
  - Aktivita je iba z časti viditeľná (prekrýva ju dialóg).
  - Aktivita aplikácie je viditeľná v móde „split screen“, ale používateľ práve používa druhú aplikáciu.
- **Stopped** – aktivita je v tomto stave, keď už nie je viac viditeľná. Používateľ spustil ďalšiu aktivitu, alebo prešiel na domovskú obrazovku.
- **Destroyed** – do tohto stavu sa aktivita môže dostať v prípade, že:
  - Bola zavolaná metóda `finish()`.
  - Používateľ sa vrátil späť do predchádzajúcej aktivity.
  - Zariadenie potrebuje uvoľniť operačnú pamäť.
  - Zmenila sa konfigurácia zariadenie (používateľ otočil obrazovku).

  Úlohou metódy `onDestroy()` je preto zmazať všetky načítané dáta a zrušiť všetky procesy na pozadí, ktoré aktivita spustila. V prípade, že používateľ otočil obrazovku, vytvorí sa nová inštancia aktivity a zavolá sa opäť metóda `onCreate()`. Celý cyklus tak začína odznova.



Obr. 2.2: Životný cyklus aktivity

[2]Stav *Running* sa v literatúre označuje aj ako *Resumed*.

Fragment

Hlavným problémom aktivít je neschopnosť bežať zároveň vedľa seba na obrazovke[3]. Vyplýva to hlavne z konceptu, akým boli navrhnuté. Na mobilných zariadeniach s menším displejom to nevadí, pretože na to ani nie je miesto. Problém však nastáva pri tvorbe používatelské rozhranie pre tablety, ktoré sa skladá z viacerých nezávislých „okien“. Príkladom môže byť aplikácia Gmail, ktorá na väčších displejoch zobrazuje zoznam mailov spolu s obsahom zvoleného mailu. Tieto „okná“ sú v skutočnosti fragmenty, ktoré vznikli práve za týmto účelom. Fragment je vždy súčasťou nejakej aktivity. Rovnako ako aktivita, tak aj fragment má svoj životný cyklus, ktorý je priamo ovplyvnený cyklom aktivity.

Intent

Intenty (zámery) sú systémové správy, ktoré upozorňujú na rôzne udalosti. Tieto udalosti so sebou nesú informáciu o akcii, ktorá sa má vykonať na ich obslúženie. Navyše, každý intent môže niesť aj pridané dáta, ktoré sú pre daný intent špecifické. Jednotlivé aktivity aplikácie nielenže môžu reagovať na intenty, ale ich môžu aj vytvárať. Takýmto spôsobom aktivity komunikujú medzi sebou.

Pri tvorbe intentu je možné dopredu určiť, ktorá aktivita má danú udalosť obslúžiť. Takýto typ správy sa nazýva *explicitný intent*. V opačnom prípade hovoríme o *implicitnom intente*. Každá aktivita môže definovať, na aké intenty je schopná reagovať. Tomuto mechanizmu sa hovorí *filter intentov*. Keď nejaká aktivita vytvorí implicitný intent, operačný systém vyberie aktivity, ktorej filter sa zhoduje s intentom. V prípade, že ich je viac, tak systém používateľovi dá na výber sa rozhodnúť, ktorá aktivita sa má použiť na dokončenie akcie. Pomocou intentov môžu komunikovať aj aplikácie medzi sebou.

Content provider

Content provider (poskytovateľ obsahu) je komponent, ktorý zabezpečuje určitú abstrakciu nad prístupom k dátam. Je to akási vrstva, ktorá oddeľuje dáta od aplikačnej logiky. Aplikácia tak nemusí vôbec riešiť spôsob, akým sú dáta uložené a kde sú uložené. Content provider taktiež umožňuje zdieľať dáta s inými aplikáciami.

V tejto súvislosti je treba spomenúť tiež *Content resolver*. Content resolver je klient, ktorý umožňuje aplikácii komunikovať s content providerom. Dáva k dispozícii tzv. CRUD rozhranie na vkladanie (create), čítanie (read), upravovanie (update) a mazanie (delete) dát. Prístup k tomuto klientovi je sprostredkovaný pomocou aktivity.

Resources

Ďalším dôležitým komponentom aplikácie sú zdroje[4] (angl. resources). Slúžia na ukladanie statického obsahu, ako sú napríklad obrázky, ikony, textové reťazce, súbory s popisom používateľského rozhrania a mnoho ďalších. Všetky tieto zdroje sú uložené oddelene od zdrojových súborov, spravidla vždy v adresári `res/`. Vďaka tomu je možné definovať rôzne návrhy používateľského rozhrania v závislosti na veľkosti obrazovky, či rôzne sady textových reťazcov pre ďalšie jazyky.

[3] Okrem prípadu, kedy sú dve aplikácie rozdelené vedľa seba – „split screen.“

[4] Niektoré literatúry používajú preklad prostriedky.

Súbor manifest

Základom každej aplikácie pre Android je tiež súbor manifest. V tomto súbore aplikácia deklaruje svoj obsah, t.j. aktivity, potrebné oprávnenia apod. Rovnako tu aplikácia definuje, ako sú jej časti prepojené so systémom, napr. filtre intentov pre aktivity.

2.3 Grafické rozhranie

S tvorbou mobilných aplikácií úzko súvisí návrh grafického používateľského rozhrania. Aplikácia môže mať obrovský počet funkcií, ale ak nemá dobre premyslené grafické rozhranie, tak tieto funkcie málokto ocení. Preto je dobré sa hneď na začiatku tvorby nových funkcií zamyslieť nad spôsobom, akým budú tieto funkcie používateľovi zobrazené.

Prvky grafického rozhrania môžeme rozdeliť podľa funkcie na:

- **widgety** – viditeľné objekty, s ktorými používateľ interaguje. V systéme Android sa označujú aj ako **Views**[5]. Patria sem tlačidlá, popisky, obrázky apod.
- **kontajnery** – slúžia na umiestnenie widgetov na obrazovke. Medzi základných predstaviteľov kontajnerov v prostredí Android radíme **LinearLayout**, **FrameLayout** a **RelativeLayout**.

Grafické rozhranie sa dá vyformovať vytváraním inštancií jednotlivých widgetov v zdrojovom kóde, keďže každý widget je vlastne trieda v jazyku Java.

Častejšie sa však stretneme s popisom rozhrania do hierarchickej XML štruktúry zvanej tiež ako *layout*. V zdrojovom texte 2.1 je možné vidieť layout, ktorý sa skladá z jedného kontajneru a z dvoch widgetov. Okrem názvu komponenty obsahuje každá entinta aj atribúty. Pomocou nich sa definujú vlastnosti grafického prvku. V texte je znázornený widget **EditText**, ktorý má okrem šírky a výšky zadefinovanú nápovedu, typ vstupu a typ tlačidla, ktoré sa zobrazí na vysúvacej klávesnici pri zadávaní obsahu.

Podpora pre RTL jazyky

V súvislosti s grafickým rozhraním je dobré pripomenúť, že existujú taktiež jazyky, ktoré sa čítajú sprava doľava. Nazývajú sa tiež aj ako RTL[6] jazyky. Typickým predstaviteľom je arabčina, či hebrejčina. Od verzie JellyBean (Android 4.2) je možné definovať layouty s atribútmi s príponou `start` alebo `end`, namiesto atribútov končiacich na `left` resp. `right` (napr. `margin_start`). Vďaka tomu môže systém prispôsobiť rozloženie jednotlivých grafických prvkov k používanému jazyku.

RecyclerView

RecyclerView radíme do skupiny kontajnerov, ktorých úlohou je efektívne zobraziť veľké množstvo dát s rovnakou štruktúrou. Typickým príkladom je zoznam emailov v aplikácii Gmail. Mailová schránka môže obsahovať stovky až tisícky správ. Ak by aplikácia vytvorila dopredu kontajner na zobrazenie náhľadu pre každú správu, tak by zariadeniu došla veľmi rýchlo pamäť. Tento problém sa snaží riešiť práve RecyclerView. Funguje tak, že dopredu vytvorí a naplní určitý počet kontajnerov na zobrazenie týchto informácií podľa

[5] View je základná trieda od ktorej sú odvodené všetky objekty grafického rozhrania.
[6] Zkratka RTL vznikla z anglického spojenia **right–to–left**.

```
<FrameLayout
    layout_width="match_parent"
    layout_height="match_parent"
    layout_margin="16dp">
    <EditText
        layout_width="match_parent"
        layout_height="wrap_content"
        layout_gravity="top"
        hint="Zadajte text"
        inputType="text"
        imeOptions="actionDone"
        layout_marginStart="8dp"
        layout_marginEnd="8dp"/>
    <Button
        layout_width="wrap_content"
        layout_height="wrap_content"
        layout_margin="16dp"
        layout_gravity="bottom|end"
        text="OK"/>
</FrameLayout>
```

Zdrojový text 2.1: Zjednodušená ukážka popisu grafického rozhrania v jazyku XML. Ukážka sa skladá z kontajnera **FrameLayout** a z dvoch widgetov **EditText** a **Button**.

veľkosti obrazovky. Postupne, ako používateľ roluje v zozname na ďalšie položky, RecyclerView recykluje tieto kontajnery a napĺňa ich novými dátami namiesto toho, aby ich znova vytváral.

ConstraintLayout

Kontajnery RelativeLayout a v niektorých prípadoch aj LinearLayout potrebujú dva priechody cez svojich potomkov na výpočet ich šírky a výšky. Každá ďalšia zmena v rozhraní vyžaduje nový výpočet (napr. zmena textu). To môže byť z hľadiska výkonu výpočtovo náročný problém, najmä ak sú tieto kontajnery zanorené v ďalších kontajneroch. Preto vznikol nový kontajner **ConstraintLayout**, ktorého cieľom je zefektívniť tieto výpočty. Umožňuje, rovnako ako RelativeLayout, usporiadať svojich potomkov relatívne voči sebe, viď [7].

2.4 Material Design

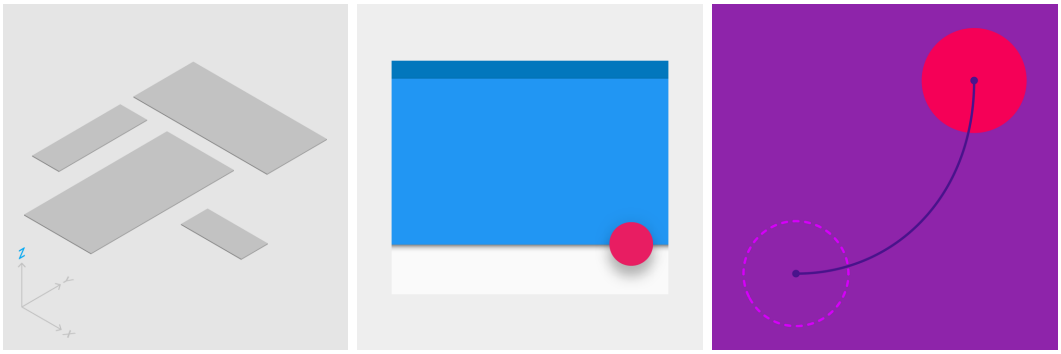
Firma Google v roku 2014 predstavila nový vizuálny jazyk zvaný *Material Design*. Jeho hlavnou myšlienkou je kombinovať princípy kvalitného dizajnu spolu s inováciami a novými technologickými možnosťami ako je dotyková obrazovka. Na obrázku 2.3 sú znázornené základné princípy tohoto jazyka, viď [4]:

1. Obrazovka sa skladá z materiálov.
2. Celé prostredie je farebné, farby sú kontrastné bez gradientov.
3. Pohyb jednotlivých vizuálnych objektov im dodáva zmysel.

Materiál Materiál môžme chápať ako vizuálny objekt jednoduchého tvaru s hrúbkou rovnou vždy jednému nezávislému pixelu[7] (obdĺžnik, štvorec, kruh, ...). Je vždy ležatý, rovnobežný s rovinou obrazovky, nikdy sa od nej neodklápa. Materiál sa neohýba, nedá sa preklopiť. Naopak, môže sa zmenšiť, zväčšiť alebo kompletne zmeniť svoj tvar. Obrazovka v prostredí materiálového dizajnu sa skladá z troch súradníc. Tretia súradnica určuje hĺbku materiálu. Virtuálny svetelný zdroj osvetľuje scénu. Toto svetlo vrhá tieň pod materiál a tým určuje jeho výšku na obrazovke.

Farby Farebná paleta materiálového dizajnu je inšpirovaná odvážnymi svetlými odtieňmi s rôznym stupňom žiarivosti. Špecifikácia takisto uvádza svoj vlastný systém farieb. V prostredí sa definuje tzv. hlavná farba, ktorá sa na obrazovke zobrazuje najčastejšie (napr. farba pozadia pre nadpis). Do kontrastu s touto farbou sa definuje tzv. sekundárna farba. Tá sa používa na zvýraznenie niektorých častí používateľského rozhrania (napr. tlačidlá). Hlavným cieľom farieb spolu s odtieňmi je naznačiť hierarchiu jednotlivých prvkov na obrazovke, dodať im význam či zdôrazniť ich stav.

Pohyb Material Design sa líši od iných vizuálnych jazykov najmä tým, že medzi svoje fundamentálne základy zaraďuje práve pohyb. Ten sa používa na opísanie priestorových vzťahov, funkcionality a zámeru v používateľskom rozhraní. Ukazuje sa, že animácie výrazne prispievajú k lepšej orientácii používateľa na obrazovke.



(a) Materiál (b) Farebné rozlíšenie (c) Pohyb na obrazovke

Obr. 2.3: Základné princípy materiálového dizajnu

CardView

Material Design definuje nový kontajner, ktorý sa v prostredí Android nazýva **CardView**. Funguje na princípe kariet. Každá karta je v podstate kus materiálu, ktorého obsahom sú informácie s rovnakým kontextom. To umožňuje lepšie naštruktúrovať zobrazované dáta používateľovi.

CardView je v podstate kontajner FrameLayout rozšírený o nové princípy z materiálového dizajnu. Má viditeľné pozadie, jeho rohy sú mierne zaoblené a zľahka vystupuje von z obrazovky (vrhá tieň).

[7] Nezávislý pixel je jednotka, ktorá predstavuje abstrakciu nad klasickým pixelom, umožňuje tak definovať pixely nezávisle na rozlíšení a veľkosti obrazovky.

Floating Action Button

Floating Action Button (ďalej v texte ako FAB) je ďalší nový komponent, ktorý vychádza z princípov materiálového dizajnu. Jeho úlohou je reprezentovať primárnu akciu, ktorú používateľ môže na obrazovke vykonať. Dobre navrhnutý grafický návrh by mal obsahovať maximálne jeden FAB. Avšak existujú aj výnimky, kedy je možné toto pravidlo porušiť, viď [4].

2.5 Dátové úložisko

Operačný systém Android disponuje mnohými rôznymi prostriedkami pre ukladanie perzistentných dát. Medzi základné určite patrí:

- **zdieľané nastavenia** – Android umožňuje ukladať dáta vo forme párov kľúč-hodnota do zdieľaných nastavení (angl. Shared Preferences). Avšak hodnoty musia byť primitívne dátové typy (boolean, float, integer, pole znakov, byte). Ich primárnym účelom je ukladať používateľove nastavenia.
- **SQLite databáza** – Systém Android obsahuje SQLite databázu vstavanú do behového prostredia. Vďaka tomu si každá aplikácia môže vytvoriť svoju vlastnú SQL databázu. SQLite databáza má nízku pamäťovú náročnosť a je pomerne rýchla.

Ďalšie možnosti

Okrem spomínaných prostriedkov systém umožňuje dáta ukladať priamo do interného úložiska, kam má každá aplikácia exkluzívny prístup. Veľkosť tohoto úložiska je však limitovaná. Ďalej aplikácia môže svoje dáta ukladať do externého úložiska, ale na to už potrebuje oprávnenie od používateľa. Ďalším možným riešením je ukladať dáta na vzdialený server.

2.6 Časté problémy pri vývoji

Táto časť je venovaná problémom, s ktorými som sa stretol pri vývoji mobilnej aplikácie. Rád by som tu zhrnul svoje poznatky, ktoré som nazbieral a naznačil možné riešenia problémov.

Problém so spätnou kompatibilitou

Z tabuľky 2.1 vyplýva, že vyše 98 % používateľov používa Android s verziou API 16 a viac, čo je v podstate deväť rôznych systémových verzií. Každá verzia prináša niečo nové. Príkladom môže byť verzia *Lollipop*, ktorá ako prvá začala natívne podporovať Material Design. Niektoré z nových vlastností často potrebujeme podporovať aj na starších verziách systému, čo je však problém.

Práve preto vznikla knižnica *Android Support Library*, ktorá čiastočne implementuje chýbajúcu funkcionalitu na starších zariadeniach. Zámerne som napísal slovo čiastočne. Použitie tejto knižnice ešte nezaručuje, že dosiahneme presne požadovaných výsledkov. Môžem spomenúť príklad s CardView. V mojom rozhraní som využil tento kontajner so zahnutými rohmi na zobrazenie klávesnice. Na zariadeniach s API 21 a viac sa tento kontajner správal tak ako by som očakával. Avšak na ostatných zariadeniach zahnuté rohy spôsobovali, že kontajner mal väčší padding než normálne. Preto som kontajner upravil pomocou Support

Library tak, aby používal kompatibilný padding. To však spôsobilo prečnievanie pozadia kontajnera. Nakoniec som bol nútený vytvoriť špeciálne pozadie so zahnutým rohom, aby pozadie neprečnievalo cez kontajner. A takýchto prípadov je viac. Dala by sa napísať celá jedna kapitola o problémoch so spätnou kompatibilitou.

Z týchto dôvodov je správne sa hneď v úvode vývoja zamyslieť, či má zmysel podporovať čo najväčší počet zariadení za cenu straty niektorých nových vlastností alebo to zmysel jednoducho nemá.

API	10	15	16 – 18	19	21 – 22	23	24 – 25
Verzia	2.3	4.0	4.1 - 4.3	4.4	5.0 - 5.1	6.0	7.0 - 7.1
Označenie	Gingerbread	Ice Cream Sandwich	Jelly Bean	KitKat	Lollipop	Marshmallow	Nougat
Podiel	0.9%	0.9%	10.1%	20.0%	32.0%	31.2%	4.9%

Tabuľka 2.1: Podiel Android verzií na trhu k 3. 4. 2017. Dáta pochádzajú z oficiálnej stránky pre Android vývojárov [2].

Zariadenia od rôznych výrobcov

Zdrojové súbory operačného systému Android sú voľne dostupné na internete[8]. Vďaka tomu má každý výrobca mobilných zariadení možnosť si systém upraviť podľa svojich potrieb. Bohužiaľ sa však stáva, že výsledkom týchto úprav sú rôzne anomálie, s ktorými vývojári pri vývoji nepočítajú. Preto je testovanie na čo najväčšom počte zariadení jediný možný spôsob, ako sa uistiť, že aplikácia sa bude správať vždy rovnako dobre.

Rôzne veľkosti obrazovky

Každé zariadenie môže mať inú veľkosť obrazovky. Toto nie je problém, ide skôr o vlastnosť mobilných zariadení. Preto je dobré si na začiatku stanoviť, pre akú veľkosť obrazovky má byť aplikácia určená. V prípade, ak má aplikácia podporovať rôzne veľkosti, je vhodné začať návrhom pre väčšie displeje. Tento návrh tak môže obsahovať väčší počet informácií, keďže návrh nie je veľmi obmedzený šírkou displeja. Následne pri návrhu na menšie displeje sú postupne odobrané nadbytočné informácie. V návrhu tak ostanú len informácie, ktoré sú v danom kontexte kľúčové. Týmto spôsobom vznikne niekoľko rôznych návrhov pre rôzne veľkosti displeja.

Android vývojárom umožňuje definovať alternatívne zdroje (angl. resources) pre rôzne zariadenia. Mne sa osvedčilo definovať tieto alternatívy na základe šírky obrazovky. Rozlišujem rozhrania pre mobilné telefóny, pre 7 palcové tablety a pre 10 a viac palcové tablety.

[8] Zdrojové súbory sú dostupné na adrese `https://source.android.com/source/downloading`.

Kapitola 3

Aplikácia na evidenciu a delenie sprepitného

Zamestnanci reštaurácií sa stretávajú s problémom spravovania svojich finančných ohodnotení, ktoré získali od spokojného zákazníka. Tieto odmeny, sprepitné, často spravujú centrálne v spoločnej kase. A to najmä v prípadoch, kedy za tok peňazí zodpovedá jedna osoba. To je niekedy príčinou ich nedôvery, či nie sú o svoju časť odmeny ukrátený. V niektorých prípadoch tento strach prechádza až k strate motivácie odvádzať svoju prácu kvalitne.

Na druhej strane zamestnávatelia sú zo zákona povinní odvádzať daň z príjmu. A sprepitné je takisto príjem, ktorý podlieha zdaneniu. V praxi sa ale toto pravidlo nedodržiava. To sa však snaží zmeniť nový zákon o elektronickej evidencii tržieb, ktorý pravidlá pre podnikateľov sprísňuje, viď [3].

Ako som už spomínal v úvode, úlohou mojej práce je vytvoriť transparentný systém na správu získaného sprepitného. Svoje riešenie som sa rozhodol implementovať pre operačný systém Android. Učinil som tak najmä preto, lebo príchodom zákona sa na českom trhu rozrástol počet pokladní s týmto operačným systémom. Výslednú aplikáciu som pomenoval ako **TipSplitter**. Názov vychádza z anglických slóv *tip* – sprepitné a *split* – deliť.

3.1 Cielová skupina

Ako už bolo spomenuté, aplikácia je primárne určená pre čašníkov. Okrem nich, do styku s aplikáciou môžu prísť tiež kuchári, s ktorými sa čašníci delia o získané odmeny. Cieľovú skupinu teda tvoria zväčša mladí ľudia, ktorí sú zvyknutí pracovať s výdobytkami modernej doby. Z hľadiska spôsobu evidencie sprepitného môžeme prevádzky rozdeliť do troch skupín:

1. Prevádzky, kde je jedna určená osoba na prácu s pokladňou. Väčšinou je to prevádzkar alebo hlavný čašník. Jeho hlavnou úlohou je kasírovať zákazníkov.

2. Prevádzky, kde čašníci zdieľajú jednu pokladňu. Každý čašník kasíruje zákazníka sám.

3. Prevádzky, kde každý čašník zodpovedá sám za prijatú hotovosť. Na konci zmeny odovzdá všetky prijaté tržby do spoločnej kasy bez sprepitného.

Moje riešenie je primárne určené pre prvé dve skupiny. Avšak nič nebráni čašníkom z tretej skupiny, aby aplikáciu používali pre svoje potreby (napr. pre lepšiu kontrolu). Taktiež v texte spomínam hlavne reštaurácie, ale aplikáciu je možné použiť aj v iných zariadeniach, kde vzniká potreba spravovať sprepitné.

3.2 Existujúce riešenia

V obchode Google Play nie je veľa aplikácií, ktoré by riešili problém s delením sprepitného a už vôbec nie aplikácie, ktoré by riešili problém s ich evidenciou. Existuje však množstvo riešení, ktoré sa zaoberajú problémom podobného charakteru. Tieto aplikácie môžeme rozdeliť do dvoch skupín:

1. Aplikácie slúžiace na výpočet sprepitného

    - *Gratuity* – prehľadná aplikácia so zaujímavým dizajnom. Používateľ si zvolí výšku účtu, počet účastníkov na platbe a výšku sprepitného v percentách. Výhodou tejto aplikácie je, že používateľ si môže zvoliť typ podniku, podľa ktorého aplikácia prednastaví ideálnu výšku sprepitného. Rovnako umožňuje používateľovi zvoliť spokojnosť v rozmedzí od nespokojný — spokojný — veľmi spokojný. Tento parameter taktiež ovplyvňuje prednastavenú výšku sprepitného.
    - *Smart Tip Calculator* – aplikácia, ktorá hneď na úvod zaujme prehľadným sprievodcom, ktorý používateľa prevedie celým grafickým rozhraním. Aplikácia taktiež ponúka možnosť sprepitné rozdeliť medzi viacerých účastníkov. Disponuje tiež možnosťou zaplatiť sprepitné nie len za seba, ale aj za ďalších ľudí.

2. Aplikácie na spravovanie spoločných výdajov a platieb

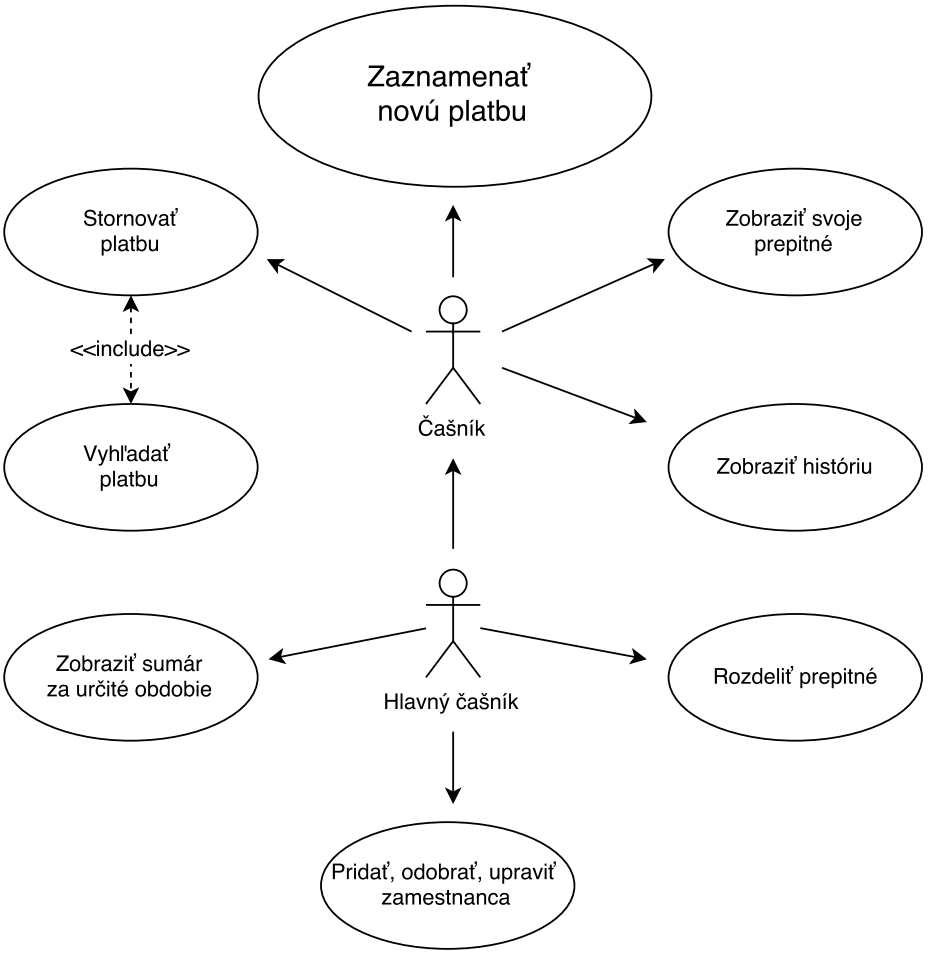
    - *Dlžníček* – aplikácie je určená pre ľudí, ktorý si medzi sebou navzájom platia účty. Jej úlohou je zaznamenať kto platbu zaplatil, aká bola hodnota účtu a za koho ďalšieho bol účet zaplatený. Ponúka používateľovi prehľad o tom kto mu dĺži peniaze. Aplikácia tiež navrhuje, kto by mal zaplatiť ďalší účet na základe dlhov. Má chytro navrhnuté používateľské rozhranie na evidenciu novej platby.
    - *Splitwise* – veľmi obľúbená aplikácia na zaznamenávanie spoločných výdajov. Aplikácia ponúka v podstate to isté čo Dlžníček s jediným rozdielom. Dlžníci sa pridávajú ručne ku každej platbe. Spôsob rozdelenia účtu sa zadáva prostredníctvom dialógu, kde má používateľ viacero možností ako platbu rozpočítať medzi dlžníkov. Na konci mesiaca aplikácia odosiela reporty o aktuálnom stave.

Spomínané aplikácie ma inšpirovali pri tvorbe môjho riešenia. Snímky obrazoviek aplikácií sú na obrázkoch v prílohe B.

3.3 Diagram prípadov užitia

Diagram prípadov užitia na obrázku 3.1 zachytáva správanie systému z pohľadu používateľa. Hlavnými aktérmi sú čašníci. Tí prídu do styku s aplikáciou najčastejšie. Ďalším aktérom je hlavný čašník resp. prevádzkar, osoba, ktorá zodpovedá za správu reštaurácie. V zásade sa tento používateľ líši od čašníka tým, že má v systéme vyššie právomoci. Inak môže vykonávať rovnaké akcie ako čašník.

Najčastejší a najpodstatnejší prípad užitia je **zaznamenať novú platbu**. V prípade, že sa čašník pomýlil alebo nastal nejaký iný problém s platbou, tak mu ju systém musí umožniť stornovať. Čašníka tiež zaujíma, aká je hodnota jeho podielu na vyzbieranom sprepitnom. Hlavný čašník môže navyše deliť sprepitné, zobrazovať reporty za vybrané obdobie a spravovať používateľov.



Obr. 3.1: Diagram prípadov užitia

3.4 Grafické používateľské rozhranie

Ešte predtým, než som začal navrhovať grafické používateľské rozhranie, som si urobil prieskum s akými pokladňami čašníci pracujú. Zistil som, že malé či stredne veľké reštaurácie a stravovacie zariadenia používajú miesto klasických pokladní práve viacúčelové tablety. Väčšinou ide o zariadenia s operačným systémom Android s dotykovou obrazovkou o rozmeroch 10 a viac palcov. Ostatné väčšie reštaurácie majú prevažne svoje vlastné riešenia navrhnuté na zakázku. Práve z týchto dôvodov som sa rozhodol svoju mobilnú aplikáciu navrhnúť pre spomínané tablety.

Počas tvorby rozhrania som svoje nápady rozoberal s čašníkmi z praxe. Tí mi obratom dali nové návrhy a postrehy na vylepšenie rozhrania. Celý proces návrhu tak prebiehal v niekoľkých iteráciách, než som vytvoril konečný výsledok. Vývoj rozhrania spomínam v kapitole 5.1 Testovanie používateľského rozhrania.

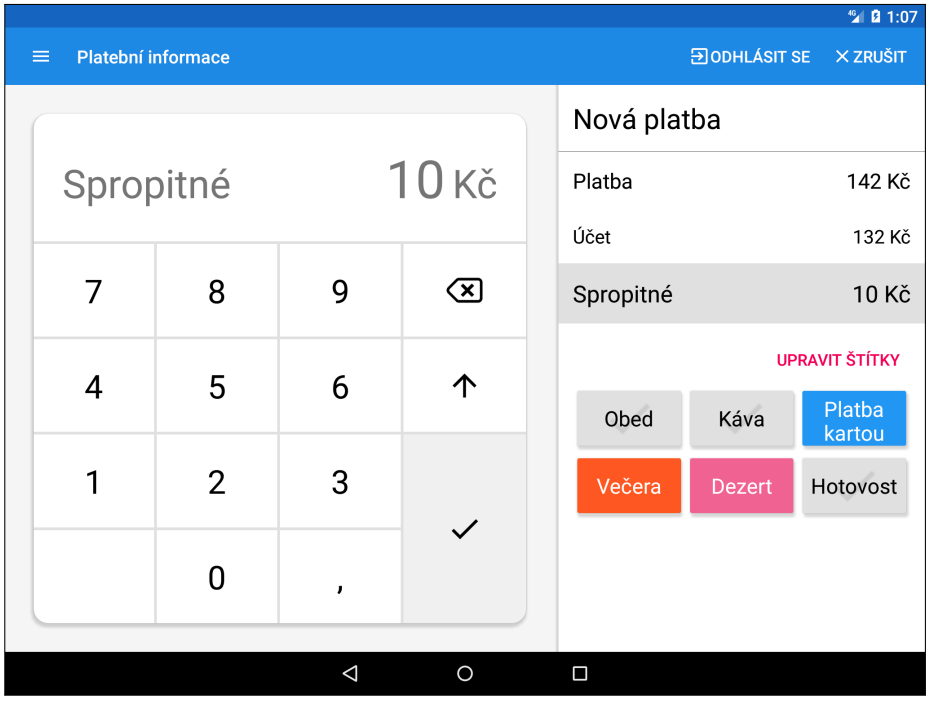
Hlavná obrazovka

Po spustení aplikácie sa zobrazí hlavná obrazovka. Jej úlohou je umožniť používateľovi zaznamenať informácie o zrealizovanej platbe. Predpokladá sa, že práve táto akcia sa bude vykonávať najčastejšie v celom systéme. Preto jej rozhranie musí byť jednoduché, intuitívne

a prehľadné. Ako je vidieť na obrázku 3.2, grafické rozhranie pozostáva z dvoch častí. Na ľavej strane obrazovky sa nachádza klávesnica, ktorá slúži na zadávanie údajov. Pravá strana pozostáva z postranného panelu, ktorého úlohou je informovať používateľa o zadaných hodnotách. Súčasťou hlavnej obrazovky je tiež navigačné menu, ktoré sa nachádza v ľavom hornom rohu. Menu umožňuje používateľovi zobraziť ďalšie možnosti aplikácie.

Vo svojom riešení požadujem, aby používateľ zadal údaje o výške účtu, o platbe, teda hodnotu, ktorú zákazník naozaj zaplatil a o sprepitnom. Celý proces zadávania musí byť rýchly. Z toho dôvodu som navrhol vlastnú klávesnicu. Používateľ zadáva údaje postupne. Najprv zadá informáciu o útrate, následne sa pomocou klávesy so šípkou nadol prepne na zadanie údaju o výške účtu. V poslednom kroku zadáva sprepitné, ktoré sa však automaticky dopočítava na základe predošlých dvoch údajov. Používateľ celú platbu potvrdí klávesou hotovo. Pohľad čašníka smeruje po celú dobu na klávesnicu. Vďaka tomu sa znižuje riziko, že urobí nejakú chybu. Ďalšou výhodou zabudovanej klávesnici je tiež fakt, že používateľ nemusí čakať pri zadávaní údajov na vysunutie systémovej klávesnice.

Ku každej platbe je možné pridať štítky. Používateľ si tak môže každú platbu personalizovať. Tieto „meta-dáta“ následne slúžia pre lepšie štatistiky v reportoch. Myšlienkou je, aby si čašníci v reportoch mohli pozrieť, na akých platbách získali najviac a najmenej sprepitného. Môže im to do budúcna pomôcť zlepšiť svoje vystupovanie pred zákazníkmi a v konečnom dôsledku si tak môžu privyrobiť na sprepitnom ešte viac.

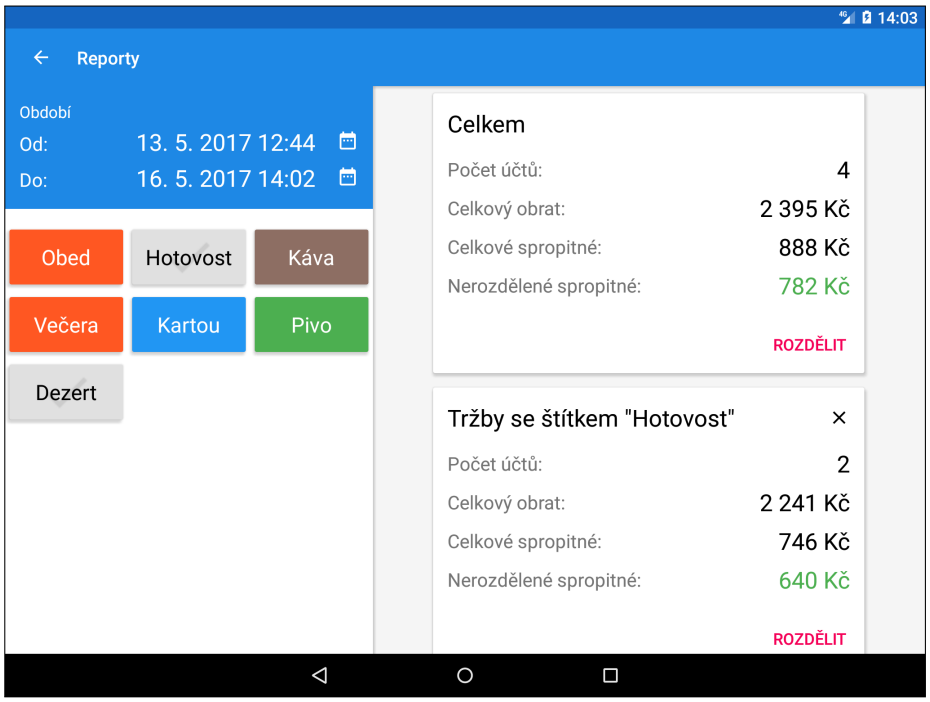


Obr. 3.2: Grafické rozhranie hlavnej obrazovky. Na ľavej strane je zobrazená zabudovaná klávesnica, ktorá slúži na zadávanie údajov. Na pravej strane je postranný informačný panel, ktorý informuje používateľa o zadaných údajoch. Obsahuje tiež tlačidlo na úpravu štítkov, ktoré sa nachádzajú v jeho spodnej časti.

Reporty

Grafické rozhranie reportov znázorňuje obrázok 3.3. Používateľ tu nájde celkový prehľad vyzbieraného sprepitného za určité obdobie. Počiatočný dátum sa automaticky doplní na čas, kedy bolo sprepitné naposledy rozdelené. Konečný dátum je aktuálny čas. Oba časové údaje je možné zmeniť pomocou dialógu na výber dátumu a času. Používateľ môže taktiež vybrať štítky, pre ktoré chce vygenerovať report.

Pravá časť obrazovky zobrazuje konkrétne reporty. Celkový report je generovaný automaticky. Po ňom nasledujú reporty podľa štítkov. Každý report obsahuje informácie o tom koľko účtov zahŕňa, aký bol obrat za vybrané obdobie, výšku vyzbieraného sprepitného a výšku nerozdeleného sprepitného za toto obdobie. Reporty so štítkom podporujú gesto *swipe* na odstránenie sumáru zo zoznamu.



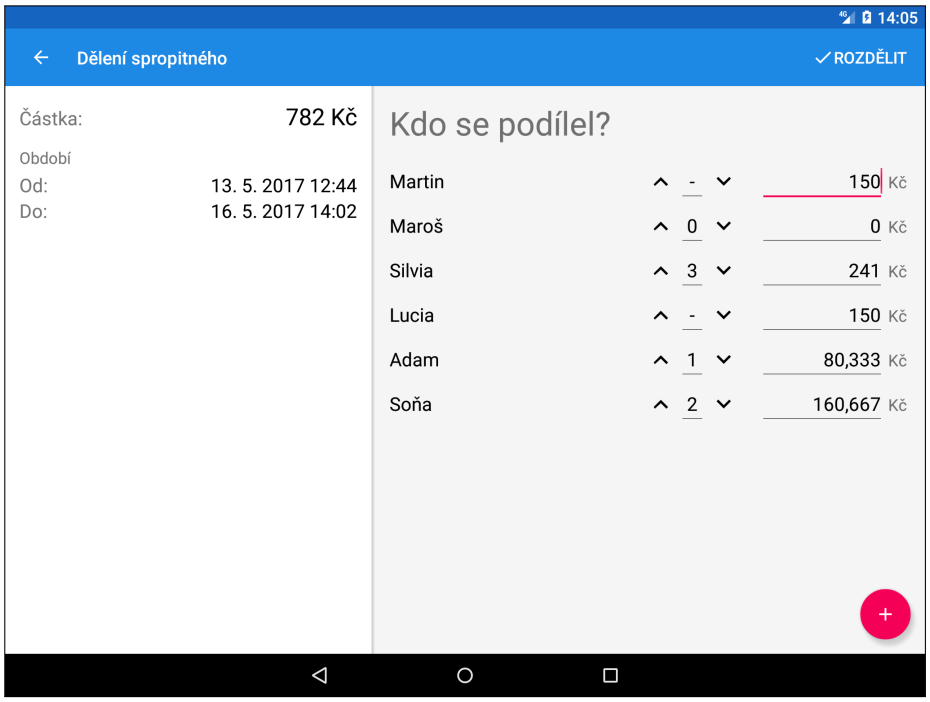
Obr. 3.3: Grafické rozhranie obrazovky s reportami. Ľavú stranu tvorí postranný panel, kde používateľ volí obdobie a štítky, pre ktoré sa majú reporty vygenerovať. Pravá strana pozostáva zo samotných reportov.

Delenie sprepitného

Každý vygenerovaný report umožňuje rozdeliť nerozdelené sprepitné. Obrazovku s delením sprepitného zachytáva obrázok 3.4. Ľavá časť obrazovky pozostáva z údajov o celkovej čiastke, ktorá je predmetom delenia a z obdobia, ktorého sa táto čiastka týka. V pravej časti sa nachádza zoznam účastníkov, medzi ktorých sa čiastka delí. Aplikácia automaticky vyberie všetkých zamestnancov, ktorý sa vo vybranom období podieľali na vyzbieranom sprepitnom. Ďalších zamestnancov je možné pridať pomocou tlačidla v pravom dolnom rohu.

Každému účastníkovi delenia môže používateľ priradiť váhu. Táto váha predstavuje podiel na získanom sprepitnom. Keď sa váha zmení, automaticky sa prepočíta podiel každému účastníkovi. Používateľ môže taktiež zadať konkrétnu hodnotu podielu každému účastní-

kovi. V takom prípade sa tento podiel odčíta od celkovej čiastky a výsledok sa prerozdelí, opäť podľa váhy, ostatným účastníkom. Každý zamestnanec si svoj podiel môže zobraziť v časti *Moje spropitné*. Pri tvorbe rozhrania na delenie sprepitného som sa inšpiroval aplikáciou *Dlžníček* (kap. 3.2).



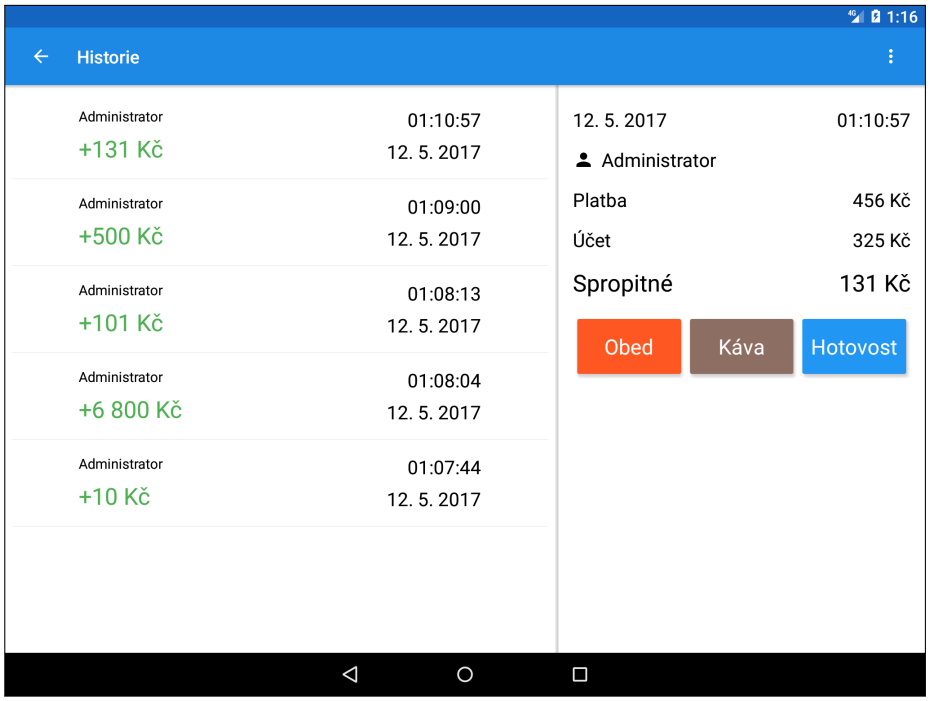
Obr. 3.4: Grafické rozhranie obrazovky s delením sprepitného. Na ľavej strane obrazovky je zobrazený údaj o čiastke, ktorá je predmetom delenia a obdobie, v ktorom bola čiastka vyzbieraná. Pravá strana obsahuje zoznam zamestnancov, medzi ktorých sa čiastka delí.

História

Grafické rozhranie histórie znázorňuje obrázok 3.5. Po otvorení sa zobrazí zoznam všetkých vytvorených platieb. Každá platba obsahuje dátum, kedy bola vytvorená, výšku sprepitného a meno zamestnanca, ktorý platbu vytvoril. Po otvorení konkrétnej platby sa v pravej časti obrazovky zobrazí jej detail. V prípade, že je niektorá platba chybne zadaná, používateľ ju môže jednoducho stornovať. Avšak môže tak učiniť jedine v prípade, že sprepitné z platby ešte nebolo predmetom delenia.

Prihlasovanie

Aplikáciu som navrhol tak, aby ju mohli používať viacerí zamestnanci z jednej reštaurácie. Každý zamestnanec má v aplikácii svoje používateľské konto. Prihlasovacia obrazovka, ktorú zachytáva obrázok 3.6, umožňuje zamestnancom sa prihlásiť buď pomocou 4-miestneho hesla alebo výberom svojho konta zo zoznamu. Akonáhle zamestnanec ukončí prácu s aplikáciou, odhlási sa. Aplikáciu nie je možné používať, kým sa neprihlási ďalší používateľ.



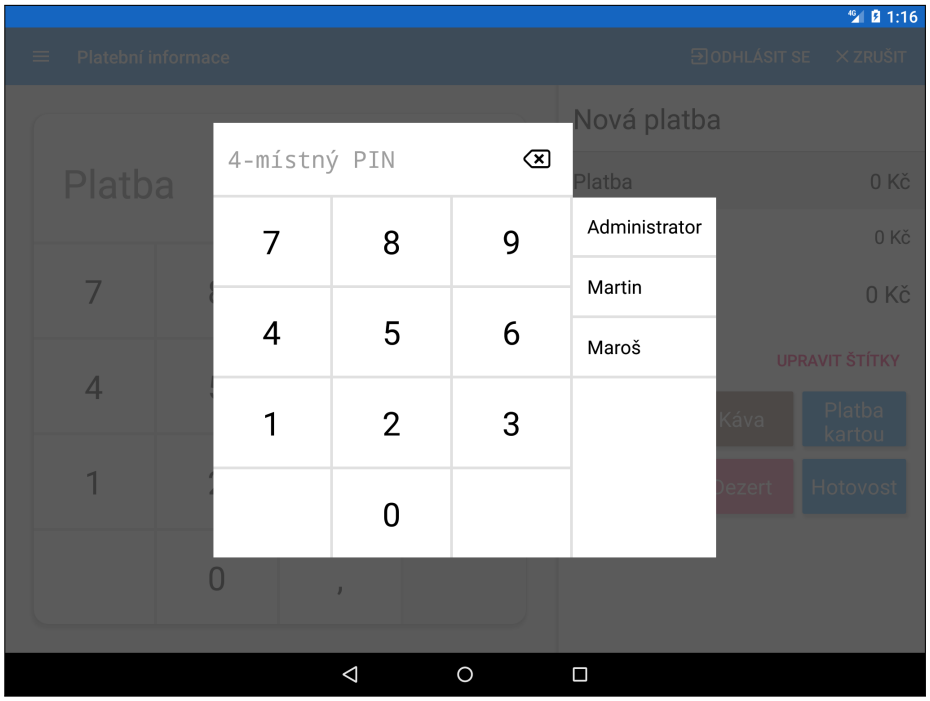
Obr. 3.5: Grafické rozhranie obrazovky s históriou platieb. Na ľavej strane sa nachádza zoznam všetkých platieb. Pravá strana slúži na zobrazenie detailu platby.

Nastavenia

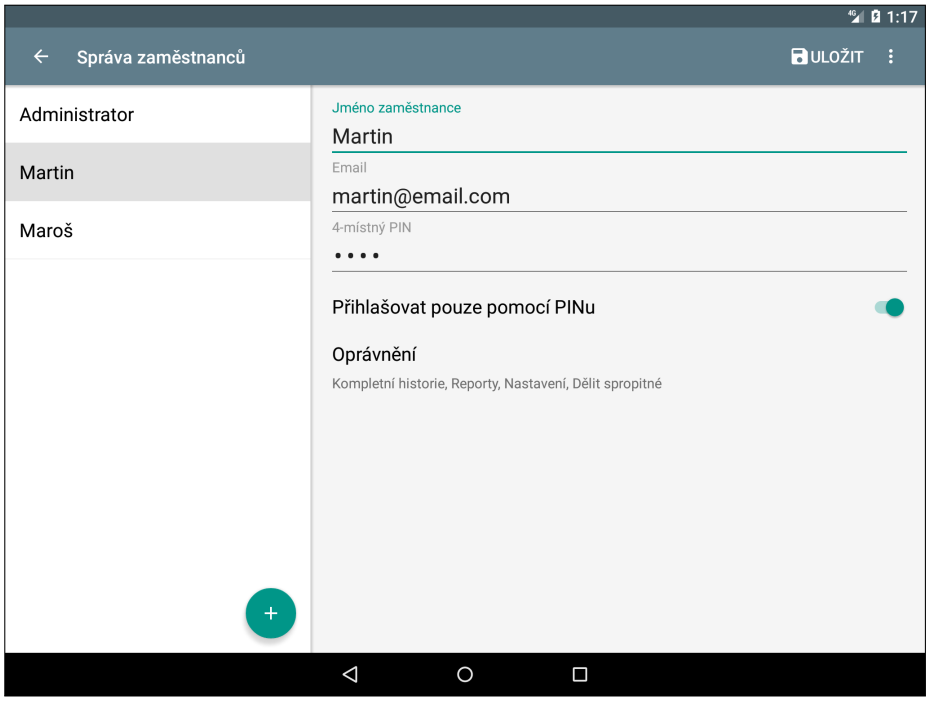
Grafické rozhranie hlavnej obrazovky s nastavením je vygenerované automaticky pomocou Android frameworku. Najpodstatnejšou voľbou je *Správa zaměstnanců*, obrázok 3.7. Ako som už spomínal v kapitole 3.3, vo svojom riešení rozlišujem dvoch rôznych aktérov – čašníka a hlavného čašníka. Preto pri tvorbe nového zamestnaneckého konta môže používateľ rozhodnúť, aké oprávnenia novému účtu pridelí. Iným riešením by bol systém rolí, ale v tomto prípade to nie je potreba, keďže používateľ má na výber len zo štyroch oprávnení:

- **Kompletná história** – Zamestnanec má právo vidieť a upravovať okrem svojich platieb aj platby svojich kolegov.
- **Reporty** – Zamestnanec má právo zobraziť si report so sprepitným.
- **Nastavenia** – Zamestnanec môže meniť nastavenia aplikácie. Tým pádom má právo pridávať ďalších zamestnancov.
- **Deliť sprepitné** – Zamestnanec môže deliť sprepitné. Toto právo úzko súvisí s právom *Reporty*.

V nastaveniach je možné meniť niektoré vlastnosti aplikácie. Osoba, ktorá má na to oprávnenie, môže zmeniť predvolený symbol používanej meny. Môže nastaviť, aby sa všetci používatelia prihlasovali iba pomocou hesla. Ďalej je tu možnosť, aby sa pri ďalšom spustení aplikácie zobrazila nápoveda, ktorá používateľovi predstaví používanie aplikácie.



Obr. 3.6: Grafické rozhranie prihlasovacej obrazovky. Rozhranie umožňuje prihlásiť sa buď pomocou hesla alebo výberom zo zoznamu používateľov.

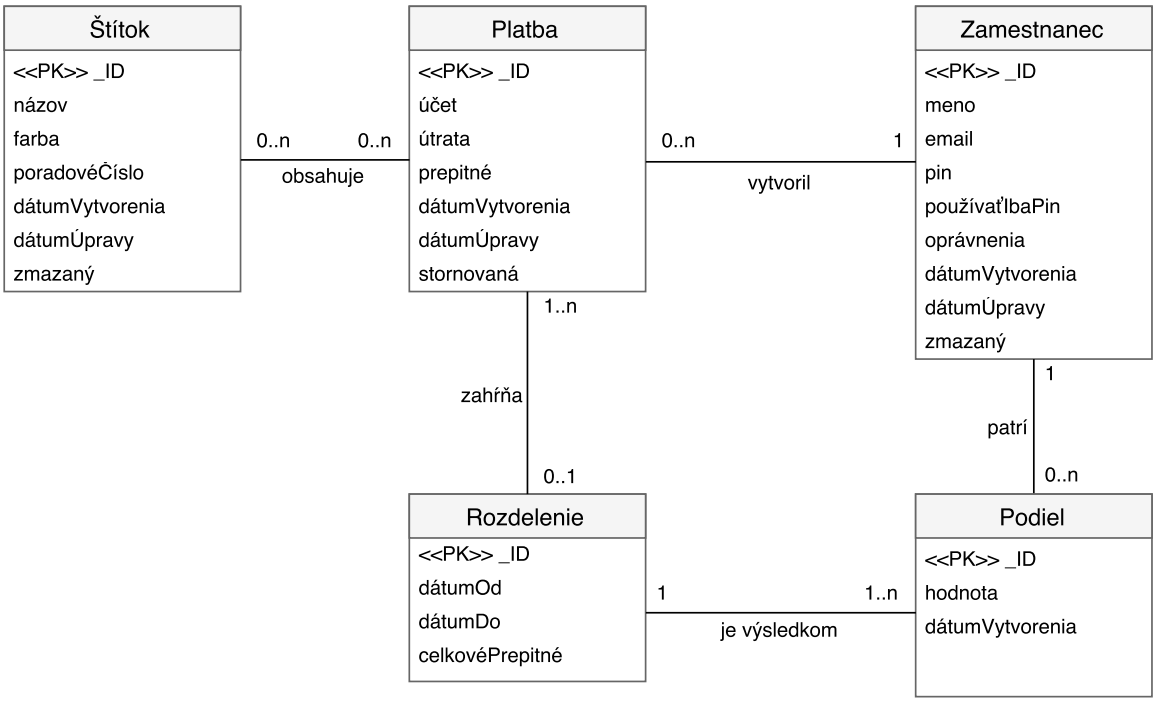


Obr. 3.7: Grafické rozhranie obrazovky so správou zamestnancov. Ľavá strana pozostáva zo zoznamu všetkých zamestnancov. Na pravej strane sa nachádza jednoduchý formulár na zadávanie informácií o zamestnancovi.

3.5 Databázový model

Vo svojej práci som sa rozhodol použiť databázu SQLite. Jej výhodou je jej rýchlosť a relatívne nízka pamäťová náročnosť. SQLite databáza je navyše vstavaná do operačného systému Android, preto ju môže použiť ľubovoľná Android aplikácia bez nutnosti zavádzať do projektu ďalšie externé závislosti. Naopak, keďže ide o veľmi odľahčenú verziu jazyka SQL, vývojár musí myslieť na niekoľko problémov s tým spojených. SQLite v prostredí Android neumožňuje mazať alebo upravovať už vytvorené stĺpce. Preto je nutné návrh databázového modelu dobre premyslieť ešte pred jeho implementáciou.

Z diagramu prípadov užitia (obr. 3.1) vyplýva, že v aplikácii je potreba ukladať najmä údaje o platbe. Okrem výšky účtu čašník zadáva tiež informáciu o útrate, čiže koľko mu zákazník naozaj zaplatil. Na základe výšky účtu a útraty aplikácia dopočíta sprepitné, ktoré sa tiež ukladá do databázy. K platbe používateľ môže nepovinne pridať niekoľko štítkov, ktoré je potrebné tiež uložiť. Moje riešenie som navrhol pre viacerých používateľov. Preto v databáze musia byť zaznamenané informácie o každom z nich. Ďalej aplikácia umožňuje deliť sprepitné medzi viacerých zamestnancov. Každý zamestnanec si následne môže pozrieť celkové sprepitné, ktoré získal za určité obdobie. Preto je nutné výsledky každého delenia zaznamenávať do databázy.



Obr. 3.8: ER diagram aplikácie TipSplitter

Relačná databáza

Konceptuálny model databázy zachytáva relačný diagram na obrázku 3.8. Sú tu naznačené vzťahy medzi jednotlivými entitnými množinami. Zaujímavým je tu najmä vzťah medzi entitami **Platba**, **Rozdelenie**, **Podiel** a **Zamestnanec**. Rozdelením sa v tomto kontexte myslí výsledok aktu, ku ktorému dochádza po delení sprepitného. Tento vzťah môžeme do bežného jazyka preložiť nasledovne: „Jedno rozdelenie môže zahŕňať niekoľko platieb,

no minimálne jednu. Na jednom rozdelení sa môže podieľať jeden až niekoľko zamestnancov. Výsledkom delenia je podiel. Každý podiel prináleží práve jednému zamestnancovi.“

Relačný model databázy vychádza z konceptuálneho modelu. Okrem tabuliek, ktoré uvádza relačný diagram, databáza tiež obsahuje tzv. „medzi-tabuľky“. Tieto tabuľky vznikli zo vzťahov M:N. Výsledná relačná databáza tak pozostáva z tabuliek **payment**, **tag**, **payment_tag**, **employee**, **employee_share**, **split**. Ďalším rozdieľom oproti relačnému modelu je inak uložený atribút `pin`. V tabuľke **zamestnanec** je tento atribút uložený ako `pinHash`. Ako už samotný názov napovedá, ide o zašifrovanú hodnotu tohto stĺpca. Viac o tomto probléme píšem v kapitole 4.

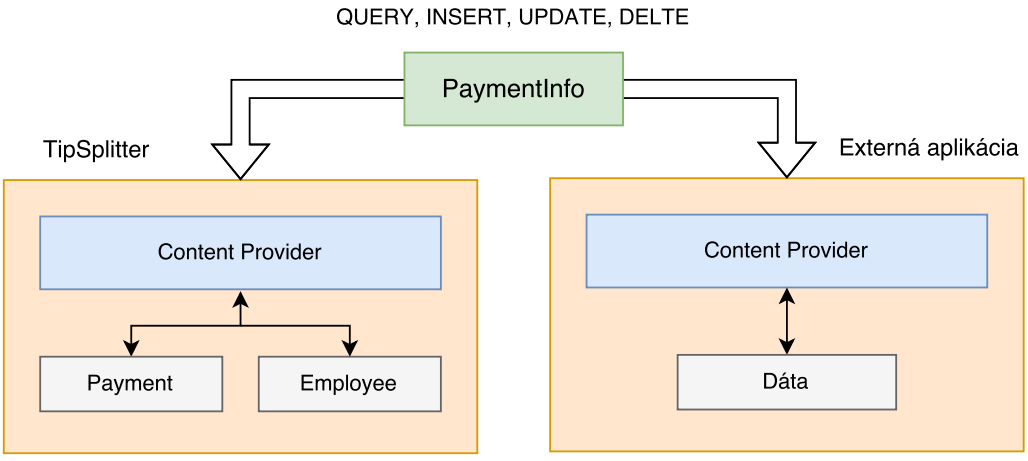
3.6 Komunikácia s inými aplikáciami

Integrácia s inými existujúcimi pokladnými systémami dáva tejto práci hlbší zmysel. Pre vývojárov takýchto aplikácií som navrhol spôsob, akým by mohli moje riešenie integrovať spoločne s ich systémom. Používateľa, ktorý má záujem evidovať a deliť sprepitné, tak odtienime od nutnosti zadávať dáta viackrát do rôznych aplikácií. Takéto riešenie po prvé urýchli celý proces zadávania informácií a po druhé minimalizuje chyby pri vstupe od samotného používateľa.

Zdieľanie dát pomocou poskytovateľa obsahu

Ako funguje poskytovateľ obsahu (angl. content provider) som už naznačil v sekcii 2.2. Aby iná aplikácia mohla komunikovať s databázou pomocou poskytovateľa obsahu, musí v prvom rade poznať presnú schému požadovanej tabuľky, t.j. názov tabuľky, názov stĺpca a jeho dátový typ. Ďalej musí aplikácia vo svojom manifeste deklarovať oprávnenia na zápis do databázy aplikácie na delenie sprepitného.

Pre tento účel som navrhol aktualizovateľný databázový pohľad **PaymentInfo**. Ide o jednoduchú schému, ktorá združuje údaje o platbe a o zamestnancovi, ktorý platbu zaevidoval. Komunikácia je naznačená na obrázku 3.9.



Obr. 3.9: Zdieľanie dát s inými aplikáciami. Aplikácie si vymieňajú dáta prostredníctvom databázového pohľadu **PaymentInfo**, ktorý združuje údaje z tabuliek **Payment** a **Employee**.

Ďalšie možnosti

Zdieľanie dát z ostatných aplikácií sa dá riešiť aj inými spôsobmi. Na strane aplikácie na evidenciu sprepitného môže existovať špeciálna aktivita na spracovanie a zápis dát. Aktivita si pomocou intent-filtru (kap. 2.2) v manifeste zadefinuje záujem prijímať intenty na zdieľanie dát. Rovnako tu zadefinuje o aký typ dát má záujem. Toto riešenie taktiež vyžaduje, aby aktivita definovala podobné komunikačné rozhranie ako v navrhovanom riešení. Jediným rozdielom je fakt, že tento spôsob nevyžaduje žiadne ďalšie oprávnenia v externej aplikácii, čo však môže prekážať používateľovi, ktorý by nevedel, že sa jeho dáta zdieľajú. Z tohoto pohľadu je čistejším riešením návrh pomocou content provideru.

Ďalším riešením je vytvoriť webové aplikačné rozhranie, s ktorým by iná aplikácia komunikovala. Dáta by boli uložené centrálne a aplikácia na delenie sprepitného by si ich sťahovala. Toto riešenie by nevyžadovalo, aby obe aplikácie boli nainštalované na rovnakom zariadení. Tento spôsob by však vniesol nutnosť implementovať vzdialený server v projekte.

Kapitola 4

Implementácia

Aplikáciu som implementoval v jazyku Java. Použil som štandardné API z Android frameworku, ktoré sa súhrne nazýva *Android Software Development Kit*. Kvôli spätnej kompatibilite so staršími verziami som taktiež využil podporné knižnice z tohoto framewroku. Svoje riešenie som implementoval vo vývojovom prostredí *Android Studio*, ktoré je prispôsobené pre tvorbu Android aplikácii. Výsledná aplikácia je dostupná pre mobilné zariadenia s operačným systémom Jelly Bean (API 15) a novším.

Táto kapitola sa zaoberá podrobnejším výkladom zaujímavých častí implementácie aplikácie. Čitateľ sa dozvie, ako vyzerá hierarchia databázových objektov v aplikácii a akým spôsobom sú uložené niektoré problematickejšie dáta v databáze (kap. 4.1). Ďalej sa dozvie, z akých častí je zložená architektúra aplikácie (kap. 4.2) a ako je implementovaná integrácia s inými aplikáciami (kap. 4.3). V závere naznačujem ďalší vývoj aplikácie (kap. 4.5).

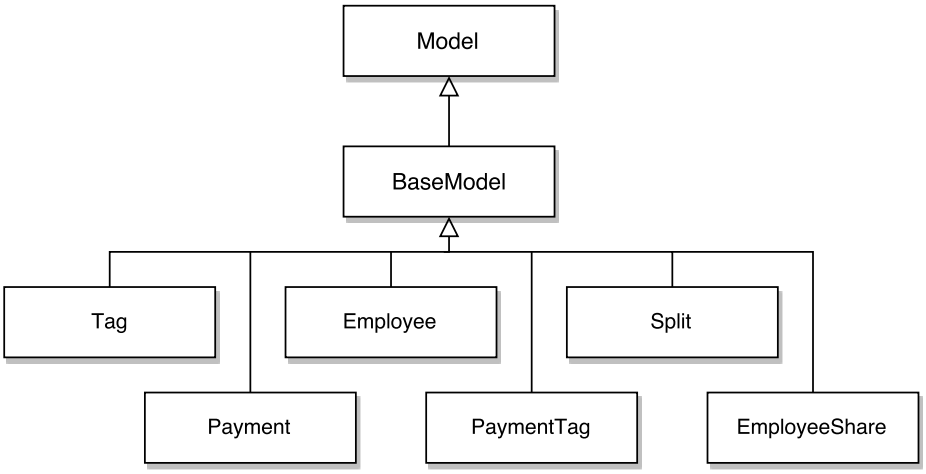
4.1 Práca s databázou

Hneď na úvod by som rád uviedol, že vo svojom projekte používam knižnicu *ActiveAndroid* na prácu s databázou. Táto knižnica za mňa rieši tvorbu nových tabuliek a taktiež spravuje content provider nad databázou. Prípadné zmeny v databázovej schéme sa tvoria pomocou SQL dotazov, ktoré ale musia byť uložené v súboroch pomenovaných podľa verzie databázy. Knižnica taktiež umožňuje mapovať jednotlivé záznamy z tabuliek do objektov v jazyku Java. Toto je hlavný dôvod, prečo som túto knižnicu použil vo svojom projekte.

Obrázok 4.1 znázorňuje hierarchiu tried databázových objektov. Každá trieda reprezentuje práve jednu tabuľku v databázovom modely. Na najvyššej úrovni je trieda **Model**. Ide o triedu z knižnice ActiveAndroid, ktorej úlohou je ukladať dáta z objektov do databázy a zase spätne namapovať dáta z databázy do objektov. Vo svojom riešení som sa rozhodol, že každý záznam v tabuľke bude navyše obsahovať časový údaj o tom, kedy bol vytvorený a aktualizovaný. Taktiež vo svojom riešení nepodporujem mazanie záznamov z databázy. Záznam, ktorý chce používateľ zmazať, sa z tabuľky fyzicky nezmaže, ale namiesto toho obsahuje informáciu o tom, že bol zmazaný. Všetky tieto informácie sú uložené v atribútoch abstraktnej triedy **BaseModel**.

Ukladanie hesiel zamestnancov

Zamestnanci sa do aplikácie môžu prihlásiť pomocou svojho hesla. Pri tvorbe zamestnanca je heslo nepovinný údaj v prípade, že v nastaveniach aplikácie nie je určené inak. Heslo zamestnanca sa ukladá do tabuľky **Employee** v zahashovanej podobe kvôli bezpečnosti.



Obr. 4.1: Hierarchia tried databázových objektov v aplikácii

Pri zisťovaní zhody uloženého hesla s heslom, ktoré zadal zamestnanec počas prihlasovania, sú obe heslá porovnávané v zahashovanej podobe. Hashovanie hesla pozostáva z nasledujúcich krokov:

1. Pri prvom spustení aplikácie sa vytvorí unikátny privátny kľúč. Ten sa vytvára iba raz za celú dobu, čo je aplikácia na zariadení nainštalovaná.
2. Vytvorí sa šifra v binárnom formáte pomocou privátneho kľúča z predchádzajúceho kroku a z hesla zamestnanca.
3. Vytvorená binárna šifra sa prevedie na reťazec čitateľných znakov pomocou Base64 kódovania.

Algoritmus používa symetrickú šifru. Celé šifrovanie zapúzdruje trieda **SecurityHelper**.

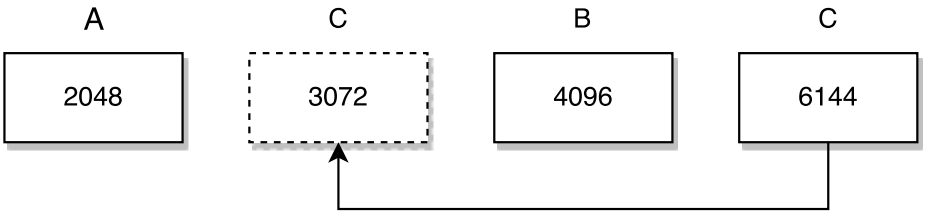
Radenie štítkov

Štítky môže používateľ v aplikácii zoradiť podľa svojho uváženia. Poradie štítkov určuje atribút `sortOrder` triedy **Tag**. Nová hodnota atribútu je po zaradení štítku rovná strednej hodnote atribútov susedných štítkov, tak ako to znázorňuje obrázok 4.2. Hodnota atribútu je vždy výsledkom celočíselného delenia. V prípade, že nová hodnota je rovná hodnote `sortOrder` susedného štítku, tak sa hodnoty atribútu všetkých štítkov znormalizujú[1] podľa vzťahu 4.1.

$$x_{n+1} = x_n + 2^{11} \qquad n \in N, \quad x_0 = 0 \tag{4.1}$$

Hodnota atribútu u nových štítkov je rovná maximálnej hodnote `sortOrder` uloženej v tabuľke, zväčšená o 2^{11}. Túto konštantu som zvolil preto, aby bola dostatočne veľká medzera medzi jednotlivými hodnotami atribútu. Vďaka tomu sa štítky nemusia často normalizovať. Konštanta je navyše násobkom čísla 2, preto bude výsledok strednej hodnoty atribútov susedných štítkov takmer vždy celé číslo. Celú prácu s radením štítkov zapúzdruje trieda **SortOrderHelper**.

[1] Štítky sú pred normalizáciou zoradené podľa hodnoty atribútu.



Obr. 4.2: Princíp radenia štítkov

Nastavenia

Aplikácia okrem databázy využíva taktiež **SharedPreferences** pre ukladanie perzistentných dát. V tomto úložisku sa ukladajú nastavenia definované používateľom, privátny kľúč, s ktorým sa šifruje heslo zamestnanca a referencia na používateľské konto prihláseného zamestnanca. Celú prácu s týmto úložiskom zapúzdruje singleton **TipSplitterPreferences**.

4.2 Architektúra aplikácie

Aplikácia je zložená z niekoľkých aktivít. Aktivity navzájom komunikujú prostredníctvom implicitných intentov, viď kapitola 2.2. Každá aktivita v manifeste definuje pomocou intent-filtru URI, na ktorom je dostupná. Pomocná trieda **IntentHelper** zapúzdruje URI každej aktivity spoločne s tvorbou nových intentov. V aplikácii je dohromady deväť aktivít, jedna aktivita pre každú obrazovku:

- **MainActivity** – hlavná obrazovka
- **ReportActivity** – obrazovka s reportami
- **HistoryActivity** – história platieb
- **SettingActivity** – nastavenia
- **EmployeeSettingActivity** – správa zamestnancov
- **LoginActivity** – prihlasovacia obrazovka
- **SplitActivity** – obrazovka s delením sprepitného
- **MyTipsActivity** – obrazovka, ktorá zobrazuje získané sprepitné
- **SendDataActivity** – aktivita, ktorej úlohou je odoslať csv dáta na mail zamestnanca

Prenášanie dát medzi aktivitami

Operačný systém Android umožňuje prostredníctvom intentov prenášať dáta vo formáte kľúč-hodnota. Hodnoty môžu byť však len primitívne dátové typy. Avšak vo svojom riešení potrebujem prenášať objekty s väčším počtom dát, napr. databázové záznamy alebo reporty medzi aktivitami **ReportActivity** a **SplitActivity**. Preto tieto objekty implementujú rozhranie **Parcelable**. To umožňuje objekty posielať v tzv. *parcelách*[2] cez intenty do ďalších aktivít. Parcela je v podstate pole primitívnych dátových typov. Implementácia rozhrania **Parcelable** obnáša:

[2]Parcela, balíček, je špeciálna trieda navrhnutá pre vyšší výkon v medziprocesovej komunikácii, viď [2].

1. Implementáciu metódy `writeToParcel()`, v ktorej je potreba všetky atribúty objektu zapísať do objektu typu **Parcel** ako primitívne dátové typy.

2. Trieda objektu musí obsahovať konštruktor s argumentom typu **Parcel**, ktorý znova vytvorí pôvodný objekt.

Fragmenty

Fragmenty sa starajú o interakciu s používateľom. Aplikácia je navrhnutá tak, aby každá aktivita obsahovala aspoň jeden fragment. Každý fragment reprezentuje jeden logický celok používateľského rozhrania, napr. **KeyboardFragment** zapúzdruje obsluhu klávesnice alebo **HistoryItemFragment** má za úlohu zobraziť zoznam všetkých platieb. Väčšina fragmentov v aplikácii sú implementované ako tzv. *retained* fragmenty. Vďaka tomu sa počas otočenia displeja nezavolá metóda `onDestroy()` na fragment, inštancia fragmentu tak otočenie prežije. Preto vo fragmentoch nemusím riešiť znovunačítavanie dát.

Ako som už spomenul, aktivita sa môže skladať z viacerých fragmentov. Tieto fragmenty však často potrebujú komunikovať medzi sebou. Práve preto vo fragmentoch používam návrhový vzor *Observer*. Pozorovaný fragment špecifikuje rozhranie, ktoré následne implementuje poslucháč. Týmto spôsobom fragmenty komunikujú medzi sebou. V aplikácii sa toto rozhranie označuje ako *listener*, napr. **KeyboardActionListener**.

Spracovanie dát na pozadí

Vykresľovanie používateľského rozhrania a interakcia s používateľom prebieha v hlavnom vlákne. Zložitejšie výpočty a načítavanie dát sa vykonávajú na vedľajšom vlákne. Android framework ponúka vlastné API na prácu s vláknami. V aplikácii využívam najmä triedy **AsyncTask** a **CursorLoader**.

Fragmenty definujú svoje vlastné statické triedy (ďalej v texte ako úlohy) pre zložitejšie výpočty. Tieto úlohy rozširujú práve triedu **AsyncTask** o svoju vlastnú logiku. Táto trieda ponúka jednoduché API na prácu v hlavnom a vedľajšom vlákne. V prípade, že úloha po dokončení výpočtu potrebuje komunikovať s fragmentom, tak takáto úloha si udržuje slabú referenciu na fragment. Ako príklad uvediem statickú triedu **LoadDataTask**, ktorej úlohou je spočítať report so sprepitným pre každý štítok a celkový report. Na konci výpočtu úloha potrebuje odoslať spočítané dáta fragmentu **OverviewReportFragment**, ktorý dáta zobrazí používateľovi. Výpočet však môže trvať príliš dlho a používateľ môže kedykoľvek odísť do inej aktivity. Ak by v takomto prípade úloha obsahovala silnú referenciu na fragment miesto slabej, tak by *garbage collector*[3] nemohol uvoľniť pamäť, ktorá bola zabraná fragmentom. Vznikol by tak „memory-leak“. Týmto spôsobom sa šetria pamäťové nároky aplikácie.

Fragmenty, ktoré potrebujú načítať dáta z databázy, využívajú Loader API[4]. Toto API definuje triedu **CursorLoader**. Tá je práve zodpovedná za načítavanie dát na pozadí. Táto trieda využíva content resolver na načítavanie dát z databázy prostredníctvom content provideru. Výsledkom načítavania dát je objekt typu **Cursor**, ktorý obsahuje referencie na záznamy z požadovaných tabuliek. Databázové objekty, ktoré spomínam v časti 4.1, implementujú metódu `loadFromCursor()`. Pomocou nej sa namapujú dáta z cursoru do objektov.

[3] Mechanizmus na uvoľňovanie nevyužívanej pamäti
[4] Loader API – https://developer.android.com/guide/components/loaders.html

4.3 Integrácia s inými aplikáciami

Ako prebieha komunikáciu medzi aplikáciami som už naznačil na obrázku 3.9. V tejto časti by som rád priblížil detaily tejto komunikácie. Celá integrácia je postavená na výmene dát medzi aplikáciami. Do svojho riešenia som preto pridal ďalší content provider **TipSplitterProvider**, ktorý je výhradne určený na komunikáciu s aplikáciami tretích strán. Keďže štruktúra **PaymentInfo** je vlastne databázový pohľad, ktorý v sebe zahŕňa stĺpce z tabuliek **Payment** a **Employee**, úlohou **TipSplitterProvider** je spracovať tieto dáta a zapísať ich do príslušných tabuliek. Provider takisto umožňuje čítať dáta z aplikácie.

Na druhej strane vývojári aplikácií tretích strán musia vo svojom manifeste deklarovať oprávnenie WRITE_TIP_SPLITTER, aby ich aplikácia mohla zapisovať dáta do aplikácie TipSplitter. Výmena dát na strane externej aplikácie prebieha prostredníctvom klienta content resolver, ktorý komunikuje s providerom. Klient však potrebuje poznať presnú štruktúru pohľadu **PaymentInfo** a URI, ktoré indetifikuje tieto dáta v provideri. Pre tieto účely som vo svojej práci vytvoril špeciálnu triedu **TipSplitterContract**, ktorá v sebe zahŕňa všetky informácie potrebné pre komunikáciu s providerom. Táto trieda je dostupná spolu s ukážkovou aplikáciou vo verejnom repozitári[5].

4.4 Použité technológie

Vo svojej práci som taktiež využil zopár knižníc, ktoré mi zjednodušili implementáciu niektorých elementárnych častí aplikácie. Všetky knižnice sú zdarma a sú verejne dostupné. Ich licencie nie sú v rozpore s ich použitím v tejto práci.

ActiveAndroid

ActiveAndroid[6] je Android knižnica na prácu s SQLite databázou. Ide o ORM[7] knižnicu, ktorá umožňuje namapovať záznamy z databázových tabuliek do objektov v jazyku Java. Knižnica taktiež ponúka Java API na prácu s SQL dotazmi.

Butterknife

Knižnica Butterknife[8] umožňuje získať referencie na widgety a kontajnery použitím jednoduchých Java anotácií. Ďalej umožňuje namapovať rôzne udalosti z používateľského rozhrania do Java metód. Táto knižnica výrazne zjednodušuje výsledný kód.

Stetho

Nástroj Stetho[9] poskytuje vývojárom prístup k databáze na mobilnom zariadení prostredníctvom internetového prehliadača Google Chrome[10]. Vývojár tak nemusí komplikovane databázu zo zariadenia sťahovať a prehľadávať ju u seba lokálne. Tento nástroj ďalej umožňuje kontrolovať prenos sieťových paketov na zariadení. Vyvíja ho firma Facebook.

[5] TipSpllitter integration – https://github.com/MartinKocour/TipSpllitter-Integration-Example
[6] ActiveAndroid – http://www.activeandroid.com
[7] Skratka pochádza z anglického spojenia „Object-relational mapping“.
[8] Butterknife – http://jakewharton.github.io/butterknife/
[9] Stetho – http://facebook.github.io/stetho/
[10] Google Chrome – https://www.google.com/chrome/browser/desktop/index.html

Crashlytics

Crashlytics[11] je nástroj na správu chýb. Poskytuje prehľadné webové rozhranie, kde sa vývojár dozvie o všetkých pádoch aplikácie. Nástroj okrem chýb zaznamenáva aj informácie o zariadení. Vývojár tak môže zistiť, na akých verziách systému Android má aplikácia problémy. Okrem pádov aplikácie nástroj umožňuje zaznamenávať aj zachytené výnimky.

StyledDialogs

Knižnica StyledDialogs[12] poskytuje jednoduché API na tvorbu Android dialógov. Vytvorené dialógy navyše spĺňajú špecifikáciu materiálového dizajnu. V projekte túto knižnicu využívam najmä na zobrazovanie rôznych chybových hlášok.

ShowcaseView

Knižnica ShowcaseView[13] poskytuje vývojárom jednoduché API na zvýrazňovanie grafických prvkov v aplikácii. Vývojár si môže zvoliť z troch tém, ktoré určujú výzor zvýrazňovania. Knižnica takisto umožňuje upraviť farby, ktoré sa pri zvýrazňovaní majú použiť. Pomocou nej som vytvoril úvodný dialóg.

Spectrum

Knižnica Spectrum[14] slúži na tvorbu farebnej palety. Ide v podstate o jednoduchý widget, pomocou ktorého si používateľ môže vybrať farbu. Vývojár pomocou jednoduchého API zadefinuje, ktoré farby sú k dispozícii a knižnica sa už postará o ich vykreslenie na obrazovku. Túto knižnicu používam v dialógu na tvorbu štítkov.

4.5 Návrhy do budúcna

V tejto časti by som rád uviedol zopár návrhov do budúcna, ktoré mi napadli pri vývoji aplikácie.

Synchronizácia medzi zariadeniami

Aplikácia je momentálne navrhnutá pre reštaurácie, kde používajú jednu pokladňu. Dáta sa nikam nesynchronizujú. Mojim návrhom do budúcna je aplikáciu upraviť tak, aby sa synchronizovali dáta na všetky zariadenia v rámci jednej reštaurácie. Čašník by sa mohol prihlásiť na ľubovoľnom zariadení a zaevidovať platbu. Reporty by obsahovali dáta zo všetkých zariadení v rámci reštaurácie. Čašník by si tiež mohol pozrieť svoje odmeny na ľubovoľnom tablete.

Na strane aplikácia by bolo treba rozšíriť každú tabuľku o stĺpec z príznakom, či už záznam bol alebo nebol synchronizovaný. Následne by sa dáta synchronizovali pomocou vzdialeného serveru. Aplikácia by navyše musela kontrolovať v rozumných intervaloch, či nenastala nejaká zmena vo vzdialenej databáze. Iným riešením je zameniť SQLite databázu

[11] Crashlytics – https://fabric.io/kits/android/crashlytics/summary
[12] StyledDialogs – https://github.com/avast/android-styled-dialogs
[13] ShowcaseView – https://github.com/amlcurran/ShowcaseView
[14] Spectrum – https://github.com/the-blue-alliance/spectrum

za inú databázu, ktorá by sa starala o synchronizáciu medzi zariadeniami. Presne k tomuto účelu by sa hodila *Firebase Realtime Database*[15] od spoločnosti Google.

Odosielanie reportov na mail

Ďalším návrhom do budúcna je odosielanie reportov na mail. Aplikácia by umožňovala odosielať celkový sumár raz za mesiac alebo v iných časových intervaloch, podľa toho ako by si to používateľ nastavil. Aplikácia by takisto umožňovala odoslať sumár s delením sprepitného každému účastníkovi, ktorý sa na delení podieľal. Zamestnanci by sa tak skôr dozvedeli, kedy bolo sprepitné rozdelené a akú čiastku získali. To by v konečnom dôsledku prispelo k väčšej transparentnosti delenia sprepitného v reštaurácii.

Pre túto funkciu by som však potreboval vytvoriť vzdialený mailový server, ktorý by odosielal tieto maily. Iným riešením by bolo odosielať maily prostredníctvom mailovej aplikácie, ktorá je väčšinou nainštalovaná na zariadení. To by však vyžadovalo interakciu s používateľom, ktorý by musel potvrdiť odosielanie každého mailu. Databázu aplikácie by som nemusel nijak zvlášť upravovať, keďže si už maily zamestnancov ukladám.

Webové rozhranie

Pre majiteľov reštaurácie alebo všeobecne pre ľudí, ktorý nemajú možnosť byť fyzicky pri zariadení s aplikáciou, by som chcel vytvoriť vzdialenú správu. Išlo by o webové rozhranie, kde by si záujemcovia mohli pozrieť ako sa darí čašníkom pri získavaní odmien. Webové rozhranie by tiež umožňovalo deliť získané sprepitné medzi zamestnancov, rovnako ako to umožňuje aplikácia. Na to by som však potreboval centralizovanú databázu, v ktorej by boli uložené dáta od všetkých používateľov aplikácie.

Diagramy v reportoch

V budúcnosti by som chcel do reportov aplikácie pridať diagramy. Tie by slúžili ako vizuálna reprezentácia dát v aplikácii. Reporty by sa tak stali o čosi prehľadnejšie pre používateľa. Aplikácia by umožňovala zobraziť vývoj získaného sprepitného v čase. Čašník by si mohol vizuálne porovnať stav získaného sprepitného podľa štítkov.

[15] NoSQL realtime databáza – https://firebase.google.com/docs/database/

Kapitola 5

Testovanie

Testovanie aplikácie prebiehalo v niekoľkých fázach. Zo začiatku som sa zameral najmä na kvalitný návrh úvodnej obrazovky. Jednotlivé návrhy som konzultoval s ľuďmi z prostredia gastronómie. V ďalšej fázy som skúsil zaujímavý experiment s inou filozofiou zadávania dát do aplikácie. V poslednej fázy som aplikáciu zverejnil v obchode Google Play. Vďaka tomu som mohol aplikáciu otestovať na rôznych zariadeniach s rôznou verziou Androidu.

5.1 Testovanie grafického rozhrania hlavnej obrazovky

S hlavnou obrazovkou príde čašník do styku najčastejšie. Preto jej ovládanie musí byť jednoduché a hlavne rýchle. Počas prvého návrhu som presne nevedel, ako by som tieto požiadavky naplnil. Z toho dôvodu som vytvoril niekoľko rôznych rozhraní, ktoré som následne testoval v reálnych podmienkach reštaurácie.

Prvý návrh

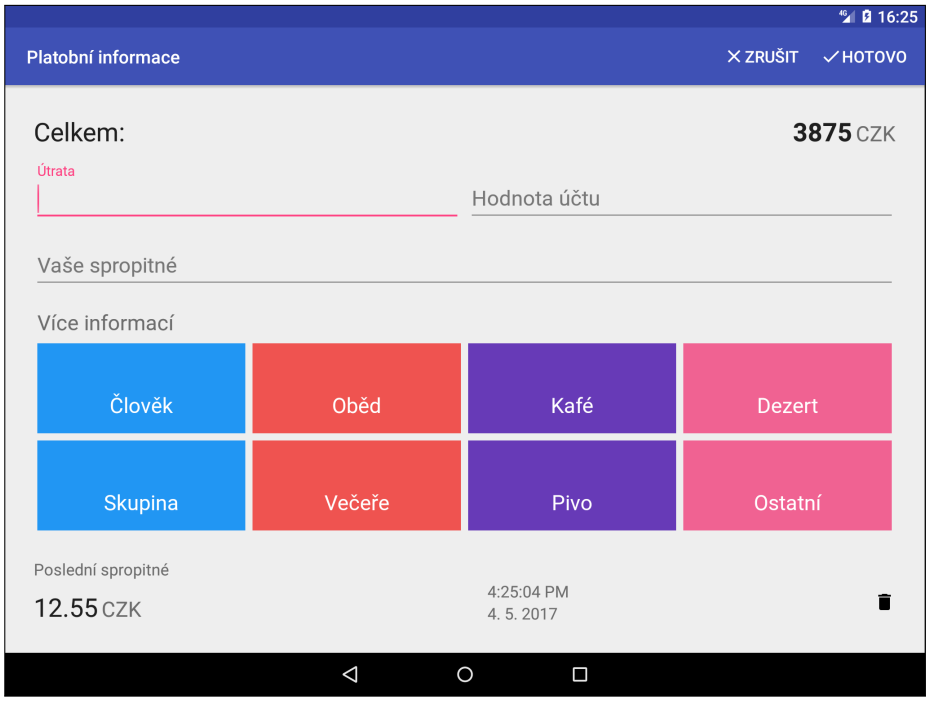
Vo svojom prvom návrhu som sa snažil čašníkovi poskytnúť čo najviac informácií. Tento návrh zachytáva obrázok 5.1. Používateľ vidí celkový prehľad vyzbieraného sprepitného, ktorý sa nachádza úplne hore. V spodnej časti obrazovky vidí poslednú vytvorenú platbu.

Ukázalo sa však, že čašníci boli z tohoto riešenia skôr zmätení. Toto rozhranie bolo pre nich neprehľadné. A hlavne sa v ňom strácala dôležitosť hlavných údajov „útrata“, „hodnota účtu“ a „sprepitné“, ktoré používateľ musí povinne vyplniť.

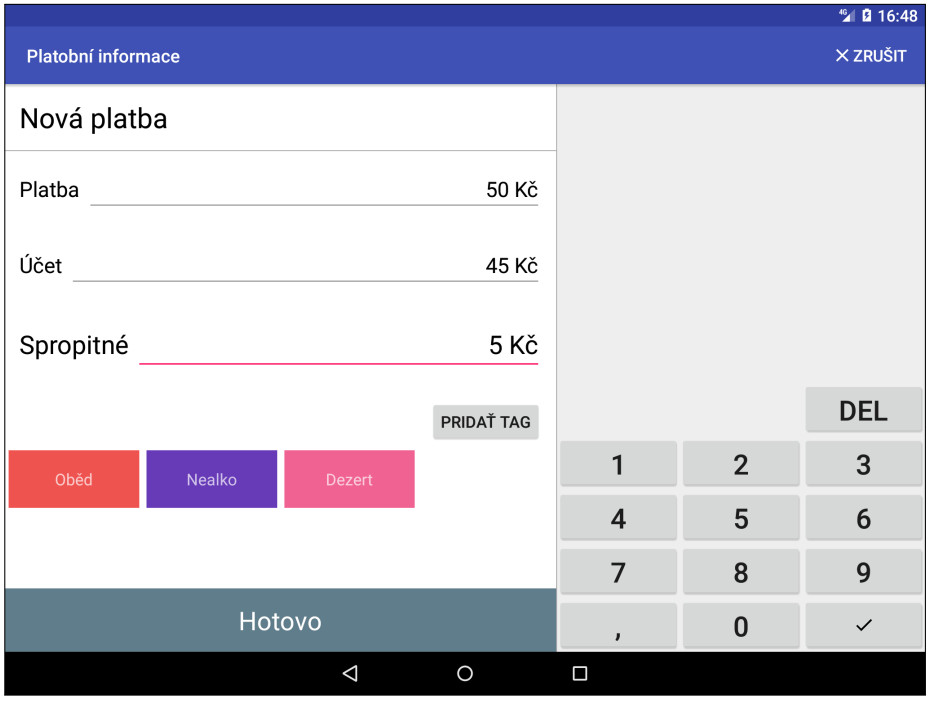
Druhý návrh

Prvá verzia neriešila veľmi problém s rýchlosťou. Vysúvacia systémová klávesnica zdržovala prácu čašníka. Preto som sa rozhodol do svojho riešenia pridať svoju vlastnú klávesnicu, ktorá je vidieť na obrázku 5.2. V tomto návrhu som navyše odstránil nepotrebné informácie, ktoré čašníka skôr rušili. V tomto návrhu sa snažím, aby sa používateľ sústredil najmä na povinné údaje.

Testovaním som však zistil, že čašníci robili chyby pri zadávaní údajov. Stávalo sa, že si čašník myslel, že už zadáva údaj o sprepitnom, no zadával stále výšku účtu. Tieto chyby robili najmä preto, lebo svoju pozornosť neustále menili z ľavej strany obrazovky na pravú a naopak. Nevnímali čo zadávajú.



Obr. 5.1: Prvý návrh hlavnej obrazovky

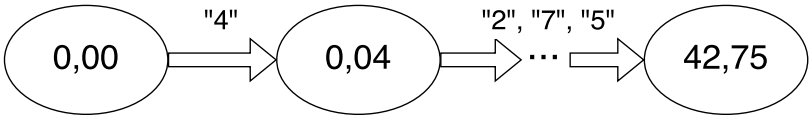


Obr. 5.2: Druhý návrh hlavnej obrazovky

Experiment s klávesnicou

Počas návštev rôznych reštaurácii som narazil aj na pokladne, ktoré fungovali na inom princípe. Vstup sa do nich zadával vrátane pevne danej desatinnej časti tak, ako to ukazuje obrázok 5.3. Rozhodol som sa teda zaexperimentovať s týmto nápadom. Z klávesnice som zmazal desatinný symbol a pridal novú klávesu „00“.

Výsledné rozhranie som následne testoval v reálnych podmienkach reštaurácie a vo svojom širšom okolí. Tento experiment sa však nestretol s veľkým pochopením. Ľudia mali problém si zvyknúť na zadávanie údajov. Celkom dlho im trvalo, kým pochopili princíp. Čašníci v reštaurácii, kde sa s takýmto zadávaním už stretli, s tým taký problém nemali. Napriek tomu som sa rozhodol tento prístup nepoužiť vo svojom finálnom riešení.



Obr. 5.3: Princíp zadávania vstupu v niektorých pokladniach

Úvodný sprievodca

V poslednej fázy vývoja som sa rozhodol svoje riešenie otestovať medzi širším spektrom ľudí. Zaujímalo ma, ako budú reagovať na výsledné používateľské rozhranie hlavnej obrazovky. Preto som vytvoril jednoduchý test. Úlohou testerov bolo zaznamenať platbu fiktívneho zákazníka do aplikácie. Dostali inštrukcie s presnou výškou hotovosti a výškou účtu. Mňa zaujímalo, ako dlho im to potrvá a či k rozhraniu potrebujú nejaký komentár. Ak potrebovali komentár, tak som ich pôvodný čas nezarátal a dal som im druhý pokus, ktorý som už zarátal. Výsledky testov znázorňuje tabuľka 5.1.

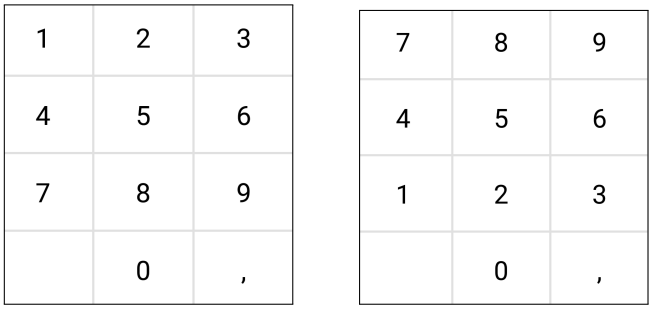
Ukázalo sa, že zadávanie údajov pomocou zabudovanej klávesnice je pre niektorých používateľov nejednoznačné. V tejto súvislosti však treba podotknúť, že testeri videli aplikáciu po prvýkrát. Akonáhle som im však ukázal, ako aplikácia pracuje, tak sa ich interakcia s rozhraním výrazne zlepšila. Práve z tohoto dôvodu som do aplikácie pridal úvodného sprievodcu. Ten prevedie používateľa celým používateľským rozhraním. Dozvie sa tak ako funguje zabudovaná klávesnica a akým spôsobom je možné využiť štítky.

veková kategória	počet osôb	priemerný čas [s]	počet osôb s komentárom	podiel osôb s komentárom [%]
18 - 25	8	12,56	1	12,5
25 - 30	4	10,81	0	0
30 - 40	6	11,23	2	33,33
40 a viac	3	17,87	2	66,67

Tabuľka 5.1: Výsledky testov grafického rozhrania hlavnej obrazovky. Výsledky ukazujú, že päť ľudí z dvadsaťjeden oslovených potrebovali vysvetliť grafické rozhranie hlavnej obrazovky. Priemerný čas strávený úlohou hodnotím ako nadmieru uspokojujúci.

Rozloženie kláves klávesnice

Počas testovania aplikácie som taktiež odhalil chybu v layoute klávesnice. Našlo sa zopár testerov, ktorým rozloženie kláves nevyhovovalo. Robili chyby pri zadávaní hodnôt, mýlili si čísla. Zistil som, že tieto chyby boli spôsobené prehodením prvého a tretieho číslicového riadku klávesnice. Rada s klávesmi „1“, „2“ a „3“ bola chybne umiestnená v hornej časti klávesnice a naopak rada s klávesmi „7“, „8“ a „9“ bola umiestnená v spodnej časti nad radou s klávesou „0“, tak ako to ukazuje obrázok 5.4. Chybné rady kláves som v ďalšej verzii prehodil.



(a) Pred úpravou (b) Po úprave

Obr. 5.4: Rozloženie kláves na klávesnici. Pred úpravou klávesnica obsahovala radu s klávesmi „7“, „8“ a „9“ v spodnej časti. Po úprave je táto rada už v hornej časti klávesnice.

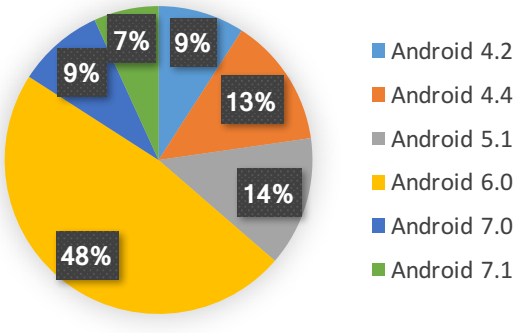
5.2 Testovanie na rôznych zariadeniach

Aplikáciu som zo začiatku testoval na tablete s Android verziou 4.2.2 (Jelly Bean), čo je vlastne minimálna verzia, ktorú aplikácia podporuje. Svoje riešenie som tiež vyvíjal a testoval v emulovanom prostredí s novšou verziou Androidu. Potreboval som však aplikáciu otestovať aj na fyzických zariadeniach s novším systémom. Preto som sa rozhodol svoje riešenie zverejniť v obchode Google Play.

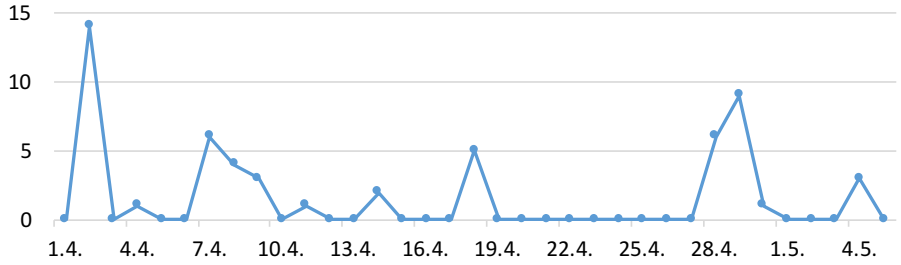
Aplikácia je momentálne k dispozícii v programe pre beta-testerov. Od jej zverejnenia si aplikáciu ľudia stiahli do viac než 20 rôznych zariadení. Graf na obrázku 5.5 znázorňuje percentuálne rozloženie zariadení s rovnakým operačným systémom. V grafe je vidieť, že aplikácia je nainštalovaná najviac na verzii 6.0 (Marshmallow). Chyby v aplikácii môžem sledovať pomocou nástroja *Crashlytics*, ktorý som popísal v časti 4.4. Graf 5.6 znázorňuje počet chýb v období od 1. 4. 2017 do 5. 5. 2017. Tieto chyby postihli dohromady 13 odlišných používateľov. Väčšinou išlo o chyby, ktoré priniesla nová verzia aplikácie. Chyby som priebežne opravoval a opravenú verziu som sa snažil čo najskôr zverejniť medzi testerov.

5.3 Zhrnutie výsledkov testovania

Testovaním používateľského rozhrania som odhalil niekoľko chýb v návrhoch hlavnej obrazovky. Cieľom bolo navrhnúť a vytvoriť čo najjednoduchší a hlavne rýchly spôsob zadávania údajov do aplikácie. Dovolím si tvrdiť, že výsledné rozhranie spĺňa tieto požiadavky. Svoju aplikáciu som navyše zverejnil v obchode Google Play. Vďaka beta-testerom môžem svoju aplikáciu testovať nielen na zariadeniach od rôznych výrobcov ale aj na rôznych verziách operačného systému Android. Aplikácia je práve vďaka ním stabilnejšia.



Obr. 5.5: Percentuálny podiel inštalácii na zariadeniach s rovnakým operačným systémom. V grafe je vidieť, že aplikácia je nainštalovaná na všetkých hlavných verziách operačného systému od verzie 4.2 (Jelly Bean).



Obr. 5.6: Graf znázorňujúci počet chýb v časovom horizonte od 1.4.2107 do 5.5.2017. V grafe je vidieť, že najviac chýb sa objavilo po prvom zverejnení beta verzie (1.4.2017).

Kapitola 6

Verzia pre mobilné telefóny

Pôvodne som svoje riešenie navrhol pre reštaurácie, kde používajú jeden spoločný tablet ako pokladňu. No ako popisujem v časti 3.1, je tu ešte jedna skupina prevádzok. Ide o reštaurácie, kde si každý čašník sám spravuje prijaté peniaze od zákazníka. Spočiatku som vo svojom návrhu s touto skupinou veľmi nepočítal. Avšak po zverejnení aplikácie medzi ľudí som bol konfrontovaný s niekoľkými názormi, že by aplikácia mala byť dostupná aj pre mobilné telefóny. Preto som sa rozhodol svoj návrh prispôsobiť aj menším obrazovkám. Tým pádom by aplikácia mohla osloviť aj čašníkov, ktorí pracujú v prevádzkach, ktoré som spomínal. Čašníci si tak sami môžu evidovať každú platbu spoločne so sprepitným na svojich telefónoch.

6.1 Používateľské rozhranie pre menšie displeje

Grafické používateľské rozhranie pre mobilné telefóny som sa snažil navrhnúť tak, aby sa veľmi nelíšilo od rozhrania pre tablety. Zo svojho riešenia som však musel odstrániť štítky. Tie na zariadeniach s malým displejom momentálne nepodporujem. No nevylučujem, že ich v budúcnosti do rozhrania zakomponujem.

Hlavná obrazovka

Z pôvodného návrhu hlavnej obrazovky som odstránil postranný panel. Používateľovi sa po otvorení aplikácie zobrazí teda len vstavaná klávesnica. Výsledný návrh grafického rozhrania je znázornený na obrázku 6.1a. Princíp fungovania klávesnice je rovnaký ako v návrhu pre tablety. Z estetických dôvodov som upravil akurát farbu nápovedy a farbu zadávanej čiastky.

Toto riešenie si však vyžiadalo aj jednu zásadnejšiu zmenu. Keďže som z pôvodného návrhu odstránil postranný panel, používateľ tak nemá možnosť pridávať k platbe štítky. To by však nemalo nijak obmedziť používanie aplikácie k účelom, pre ktoré bola navrhnutá.

Reporty a delenie sprepitného

Grafické rozhranie reportov pre menšie obrazovky znázorňuje obrázok 6.1b. Pôvodný ľavý panel, kde sa vyberá obdobie, som presunul do hornej časti obrazovky. Používateľ nemôže vygenerovať ďalšie reporty podľa štítkov, keďže ich nemožno zadať pri zaznamenávaní novej platby v hlavnej obrazovke. Používateľ tak vidí iba celkový report za vybrané obdobie.

Podobne som upravil aj obrazovku s delením sprepitného. V hornej časti používateľ nájde informácie o výške sprepitného, ktoré sa má rozdeliť spolu s obdobím, kedy bolo

sprepitné vyzbierané. V dolnej časti obrazovky sa nachádza zoznam účastníkov, ktorý si medzi sebou majú rozdeliť sprepitné.

História a správa zamestnancov

V prípade grafického rozhrania histórie a správy zamestnancov som zvolil iný spôsob. Pôvodný návrh rozhrania histórie sa skladá z výberu konkrétnej platby a z jej detailu. Na menších displejoch som sa rozhodol tento návrh rozdeliť do dvoch obrazoviek. V prvom kroku používateľ vyberie konkrétnu platbu a následne sa mu zobrazí jej detail. Tento návrh zachytávajú obrázky 6.1c a 6.1d.

Podobne som upravil aj grafické rozhranie správy zamestnancov. Používateľ najprv vyberie konkrétneho zamestnanca alebo vytvorí nový účet. Následne sa mu zobrazí obrazovka s detailom o zamestnancovi, resp. prázdny formulár. Toto rozhranie zobrazujú obrázky 6.1e a 6.1f.

6.2 Implementácia

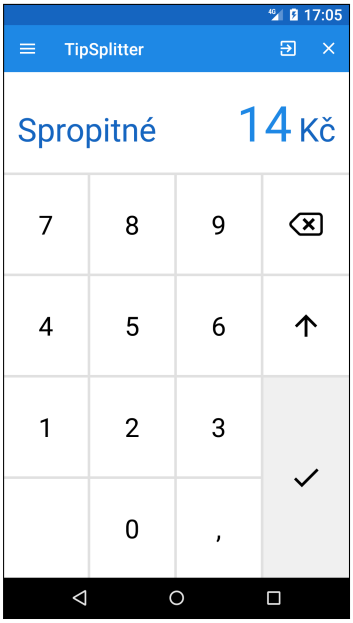
Podľa šírky obrazovky rozlišujem grafické používateľské rozhranie pre zariadenia so šírkou menšou než 480 dp a zariadenia s väčšou šírkou. Aby som nemusel upravovať zdrojový súbor hlavnej aktivity, tak som do jej layoutu pre menšie displeje pridal tiež postranný panel. Ten je však po celú dobu skrytý.

Kvôli štítkom, ktoré nepodporujem na menších displejoch, som zadefinoval premennú `tags_visibility`. Tá nadobúda hodnôt GONE pre displeje so šírkou menšou než 480 dp alebo VISIBLE pre všetky ostatné. Používam ju vo vnútornom layoute, ktorý zobrazuje štítky v reportoch. Túto premennú som musel zadefinovať aj preto, lebo vnútorný layout využívam pre všetky veľkosti obrazovky.

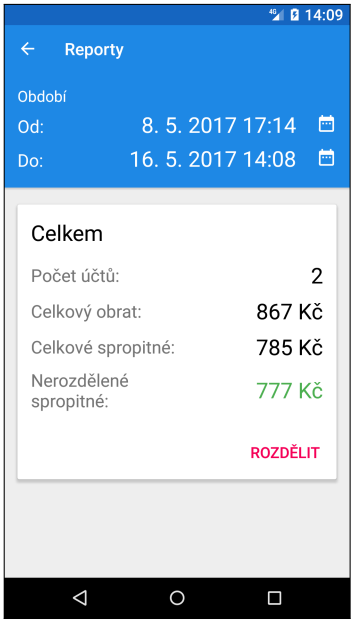
Najväčšie zmeny si vyžiadali aktivity **HistoryActivity** a **EmployeeSettingsActivity**. Layout aktivity HistoryActivity pre menšie obrazovky obsahuje kontajner `FrameLayout` namiesto `LinearLayout`. Vďaka tomu sa prekrýva zoznam platieb spolu s detailom platby, ktorý je v popredí, no nie je vidieť v počiatočnom stave. Okrem layoutu som musel upraviť tiež zdrojový súbor aktivity. Pomocou premennej `two_pane` aplikácia vie, či má zobraziť oba fragmenty. Na malých obrazovkách sa zobrazuje iba jeden fragment.

V aktivite som musel vyriešiť tiež problém so spätnou navigáciou. V momente, keď sa používateľ chce vrátiť späť, musí aplikácia rozlíšiť, či odchádza z aktivity do inej aktivity alebo chce len zatvoriť fragment. Tento problém sa rieši tak, že aplikácia zachytáva stlačenie spätného tlačidla pomocou metódy `onBackPressed()`. Jej chovanie je definované tak, že ak používateľ má pred sebou fragment s detailom platby, tak stlačením tlačidla späť sa vráti do zoznamu platieb. Inak sa zavolá metóda `onFinish()`, ktorá zničí aktivitu a používateľa vráti do predchádzajúcej aktivity. Podobným spôsobom som upravil aj aktivitu **EmployeeSettingsActivity**.

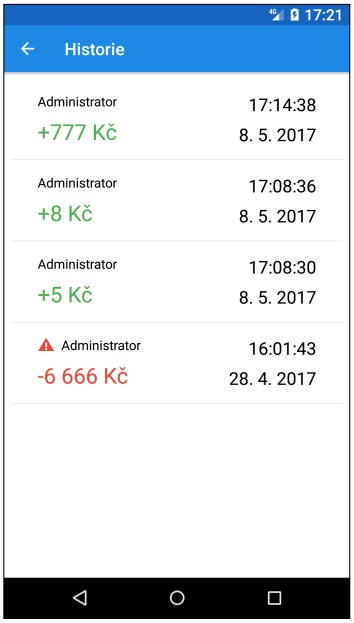
V aplikácii bolo tiež nutné upraviť prihlasovaciu obrazovku. Tá na malých displejoch neobsahuje vstavanú klávesnicu. Namiesto nej som použil widget **EditText**. Používateľ tak zadáva 4-miestny PIN pomocou systémovej klávesnice.



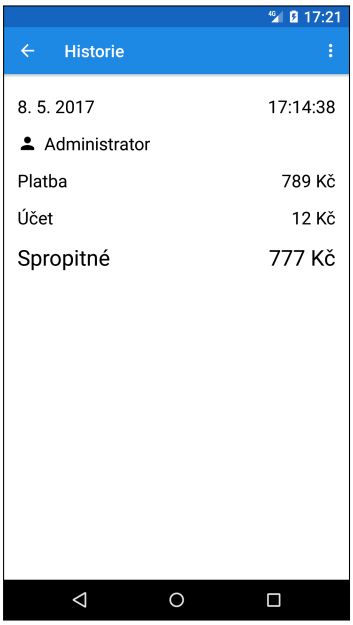
(a) Hlavná obrazovka



(b) Reporty

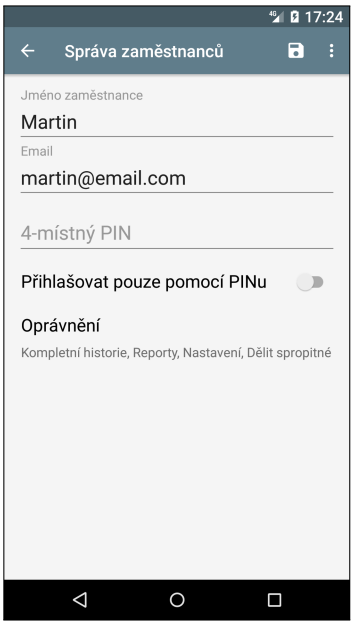


(c) Zoznam platieb



(d) Detail platby

(e) Zoznam zamestnancov



(f) Detail zamestnanca

Obr. 6.1: Grafické rozhranie pre mobilné telefóny

Kapitola 7

Záver

V rámci tejto práce sa podarilo vytvoriť Android aplikáciu pre tablety, ktorá slúži na správu sprepitného v reštaurácii. Pri riešení som preštudoval problematiku tvorby mobilných aplikácií v prostredí Android. Celkom podrobne som ju v práci opísal, pretože som to potreboval pri návrhu a implementácii svojho riešenia. Zároveň som sa musel zoznámiť s problematikou správy a delenia sprepitného v reštauráciách. Zistil som, že každá reštaurácia používa svoj vlastný systém delenia. Vo svojej práci som navrhol niekoľko variant používateľského rozhrania pre tablety. Tieto varianty som niekoľkokrát testoval v reálnych podmienkach reštaurácie ale i vo svojom okolí. Podarilo sa mi dosiahnuť riešenie, ktoré je použiteľné a zároveň spĺňa špecifikácie moderného dizajnu. Svojím používateľom ponúka nielen evidenciu finančnej odmeny, ale aj jej transparentné delenie. Navyše som vytvoril spôsob, akým by bolo možné moju aplikáciu integrovať s iným existujúcim pokladným systémom. Taktiež som vo svojej práci navrhol a implementoval grafické používateľské rozhranie pre mobilné telefóny, ktoré majú menší displej než tablet.

Táto práca ma obohatila o skúsenosť s vývojom moderných Android aplikácii. Taktiež ma naučila systematickému prístupu k vývoju, nasadeniu a údržbe softwaru. Prekvapilo ma, že neexistuje jednoduchá aplikácia, ktoré by pomáhala čašníkom riešiť problém s evidenciou sprepitného. Preto mám v pláne v najbližšom období aplikáciu ponúknuť výrobcom pokladných systémov, aby ju zaintegrovali do svojho riešenia. Výsledok tejto práce som sa rozhodol zverejniť v obchode Google Play pod názvom TipSplitter.

Literatúra

[1] *ART and Dalvik*. Google Inc., [Online; navštívené 12.04.2016].
URL https://source.android.com/devices/tech/dalvik/

[2] *Introduction to Android*. Google Inc., [Online; navštívené 23.04.2017].
URL https://developer.android.com/guide/index.html

[3] *Kdo a jaké tržby eviduje*. Finanční správa, [Online; navštívené 19.04.2017].
URL http://www.etrzby.cz/cs/kdo-a-jake-trzby-eviduje

[4] *Material design guidelines*. Google Inc., [Online; navštívené 17.04.2017].
URL https://material.io/guidelines/

[5] Chauhan, G. M. S.; Anbalagan, C.: *Strategy and Trends in B2B Business: Opportunities and Challenges - A Global Prospective*. *IFSMRC AIJRM*, ročník 02, č. 03, 2014, ISSN 2308-1341.

[6] Grant, A.: *Android 4: průvodce programováním mobilních aplikací*. Brno: Computer Press, první vydání, 2012, ISBN 978-80-251-3782-6.

[7] Murphy, M. L.: *The Busy Coder's Guide to Android Development*. United States: CommonsWare, Marec 2017, ISBN 978-0-9816780-0-9.

Prílohy

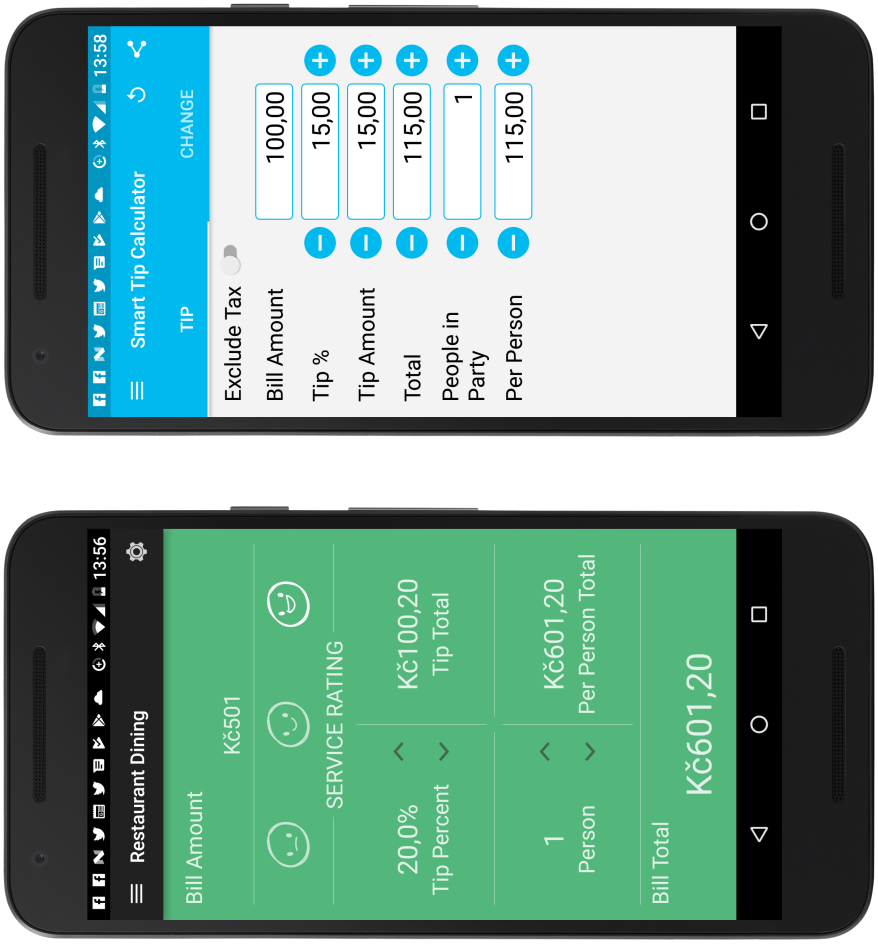
Príloha A

Obsah CD

- `android/` – adresár so zdrojovými súbormi Android aplikácie
- `latex/` – adresár so zdrojovými TEX súbormi techinckej správy
- `xkocou08-TipSplitter.pdf` – technická správa k bakalárskej práci
- `2017-xkocou08-TipSplitter.mp4` – demonštračné video k aplikácii
- `2017-xkocou08-TipSplitter.xml` – XML popis videa
- `TipSplitter.apk` – inštalačný balík aplikácie (APK)
- `plagat.pdf` – propagačný plagát aplikácie
- `README.txt` – inštrukcie k obsahu CD

Príloha B

Snímky obrazoviek existujúcich riešení

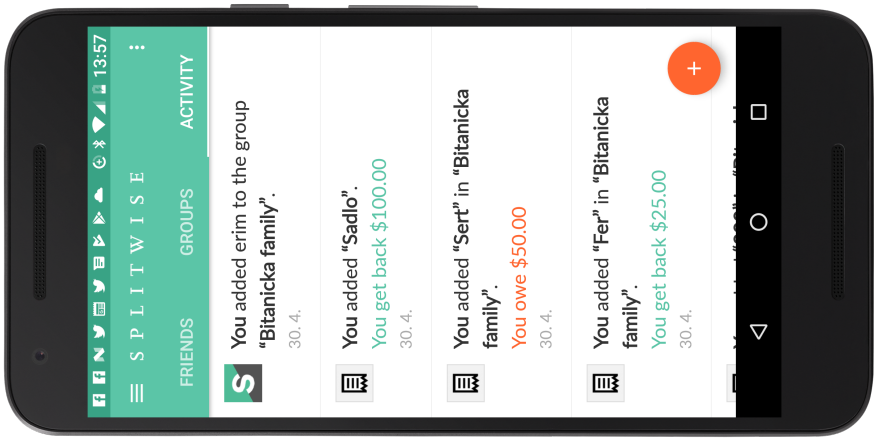


(b) Smart Tip Calculator[2]

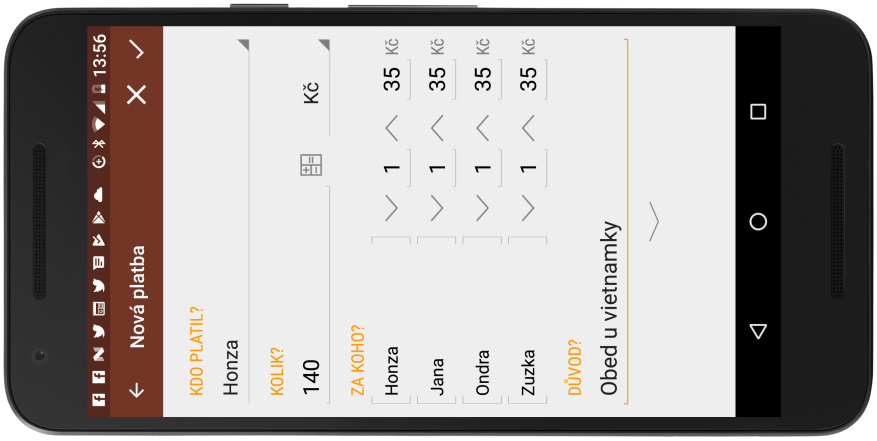
(a) Gratuity[1]

Obr. B.1: Aplikácie na výpočet sprepitného. Pomáhajú používateľovi vypočítať výšku sprepitného pre obsluhu.

[1] viď https://play.google.com/store/apps/details?id=com.mrengineer13.gratuity.free
[2] viď https://play.google.com/store/apps/details?id=com.dnbitstudio.smarttipcalculator



(b) Splitwise[4]



(a) Dlžníček[3]

Obr. B.2: Aplikácie na evidenciu dlhov. Umožňujú spravovať dlhy a výdaje medzi kamarátmi.

[3]viď https://play.google.com/store/apps/details?id=cz.destil.settleup
[4]viď https://play.google.com/store/apps/details?id=com.Splitwise.SplitwiseMobile

Príloha C

Google Play

Obr. C.1: Odkaz na aplikáciu **TipSplitter** v obchode Google Play[1]

[1] TipSplitter – https://play.google.com/store/apps/details?id=com.kocopepo.tipsplitter