VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

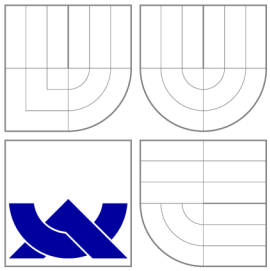
INTELIGENTNÍ NÁKUPNÍ LÍSTEK

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

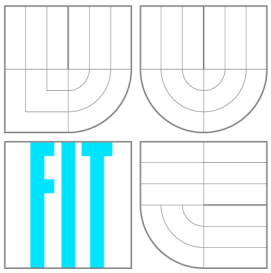
AUTOR PRÁCE
AUTHOR

Bc. MILAN DOUBEK

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

INTELIGENTNÍ NÁKUPNÍ LÍSTEK

INTELLIGENT SHOPPING LIST

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MILAN DOUBEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. IGOR SZÖKE, Ph.D.

BRNO 2012

Abstrakt

Tato práce se zabývá tvorbou unikátní aplikace pro správu nákupních lístků, při jejímž vývoji bylo využito nejnovějších technik a principů používaných u začínajících projektů, tzv. startupů. Veškeré hypotézy a úvahy byly testovány na vzorku počátečních uživatelů zvaných early adopters. Na základě jejich zpětné vazby bylo rozhodováno o dalším směru vývoje. Výsledkem práce je mobilní aplikace pro operační systém Android, která je veřejně dostupná na Google Play marketu, a dvě komponenty, o které bude tato aplikace v budoucnu na marketu rozšířena. Tou hlavní je inteligentní řazení položek na nákupním lístku podle modelu obchodu, který je vytvářen z předchozích nákupů v tomto obchodě. Druhou je pak webová aplikace, přes kterou je možné do mobilního zařízení posílat nové nákupní lístky.

Abstract

This thesis deals with creating of unique shopping lists management application and we used the newest startup techniques and principles during its development. All our hypotheses were tested by early adopters and the new courses of development were based on their feedback. The result of this thesis is a mobile application for Android operating system which is placed on Google Play market and two its components which will extend the application on the market. The main component is inteligent sorting of items on the shopping list by the supermarket model, which is created from last purchases in this supermarket. The second one is web application enabling us send new shopping lists to the mobile device.

Klíčová slova

Android, mobilní aplikace, webová aplikace, startup, SQLite, FTS3, early adopters, framework Nette, jazykový model, n-gram model

Keywords

Android, mobile application, web application, startup, SQLite, FTS3, early adopters, Nette framework, language model, n-gram model

Citace

Milan Doubek: Inteligentní nákupní lístek, diplomová práce, Brno, FIT VUT v Brně, 2012

Inteligentní nákupní lístek

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Igora Szökeho, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Milan Doubek
22. května 2012

Poděkování

Chtěl bych velmi poděkovat svému vedoucímu Ing. Igoru Szökemu, Ph.D. za nadstandardní vedení mé práce, motivační konzultace a celkově výborný přístup. Dále pak všem, kteří mě při práci podporovali.

Obsah

1 Úvod **3**

2 Metodika vývoje projektu **4**
2.1 Téma práce 4
2.2 Startupy a jejich podpůrné metody 4
2.3 Early adopters 5
2.4 Shrnutí 5

3 Mobilní aplikace **7**
3.1 Typy mobilních aplikací 7
3.2 Operační systém Android 8
3.3 Konkurenční aplikace 9
3.3.1 Out of Milk 9
3.3.2 Fivefly Shopping list 9
3.3.3 Mighty Grocery Shopping 10
3.3.4 Závěrečné zhodnocení konkurence 11
3.4 Vývojové nástroje a prostředí 11
3.5 Struktura Android projektu 12
3.6 Architektura operačního systému 13
3.7 Architektura frameworku uživatelského rozhraní 14
3.8 Komponenty aplikace 15
3.9 Uložení perzistentních dat 16
3.10 SQLite databáze 16
3.11 SQLite FTS3 rozšíření 17
3.12 Práce s vlákny 18
3.13 Systém vykreslování 19
3.14 Optimalizace grafického rozhraní pro různá zařízení 19
3.15 Návrh datové struktury 20
3.16 Návrh vzhledu aplikace 22
3.17 Shrnutí 23

4 Webová aplikace **24**
4.1 K čemu slouží webový framework 24
4.2 Webový framework Nette 24
4.3 Architektura Nette 25
4.4 Struktura Nette projektu 25
4.5 Životní cyklus presenteru 25
4.6 Návrh propojení s mobilní aplikací 26

4.7 Uživatelské účty . . . 27
4.8 Návrh datové struktury . . . 28
4.9 Shrnutí . . . 29

5 Inteligentní řazení položek **30**
5.1 Jazykový model . . . 30
5.2 Oceňování n-gramového modelu . . . 31
5.3 Vyhodnocení lístku modelem . . . 32
5.4 Složitost algoritmu . . . 33
5.5 Shrnutí . . . 34

6 Implementace a vyhodnocení **35**
6.1 Získávání zpětné vazby . . . 35
6.2 Mobilní aplikace . . . 36
6.3 Vestavěná databáze položek . . . 37
6.4 Webová aplikace a její zakomponování . . . 38
6.5 Inteligentní řazení položek . . . 41
6.6 Vyhodnocení dat z databáze . . . 43
6.7 Shrnutí . . . 43

7 Závěr **45**

A Obsah CD **48**

Kapitola 1

Úvod

V této diplomové práci si dávám za cíl vytvoření inovativní aplikace pro mobilní telefony spojené s webovým rozhraním, která bude usnadňovat lidem život a šetřit jejich čas. Za smysl celé práce považuji vytvoření softwaru, který nebude vyvíjen pouze v rámci fakulty pro účely diplomové práce, ale bude schopen reálného nasazení v praxi a využití lidmi po celém světě. Chci využít nejnovějších metod a principů, které se používají při vývoji začínajících inovativních projektů, tzv. startapů, pro které je typická snaha o úspěšné zavedení, získání uživatelů a v neposlední řadě také o zisk. Od toho se také odvíjejí použité metody vývoje, testování a získávání zpětné vazby. Nepůjde tedy pouze o běžnou školní práci, ale bude zde kladen důraz na její finální výstup, který by měl být využitelný pro reálné uživatele. Více se čtenář dozví v kapitole 2, kde bude problematika probrána podrobněji. Zároveň zde bude představeno téma práce, kterým je *Inteligentní nákupní lístek*.

Vzhledem ke zvolenému konceptu bude diplomová práce rozdělena do pěti kapitol. Jak již bylo předesláno v předchozím odstavci, ve druhé kapitole bude popsáno téma práce a metodika jejího vývoje. Kapitoly 3 a 4 pak představí dvě ústřední části této práce, a to mobilní a webovou aplikaci. Zejména pak návrh a teoretický základ, který bylo třeba pro jejich vytvoření nastudovat. V kapitole 5 pak bude představena inovativní funkce inteligentního řazení položek na nákupním lístku, která byla v rámci projektu vytvořena. Opět se bude skládat z teoretického rozboru a návrhu. Výsledná podoba projektu, vzniklá spojením těchto částí, je popsána v kapitole 6. Zde jsou také zhodnoceny dosažené výsledky a funkčnost jak pro jednotlivé části, tak pro celkovou práci.

Kapitola 2

Metodika vývoje projektu

Jak bylo uvedeno v úvodu práce, tato kapitola čtenáři přiblíží výběr tématu vytvářených aplikací a vlastně celé diplomové práce. Dále zde bude popsán termín **startup** a metody, které s ním souvisí. Zaměříme se zejména na testování předpokladů a získávání zpětné vazby od uživatelů.

2.1 Téma práce

Proč bylo vybráno zrovna téma nákupního lístku? Důvod je jednoduchý. Každý musí někdy nakupovat, což nám dává velké množství potencionálních uživatelů. Ovšem zapamatovat si všechny položky, o kterých v průběhu času zjistíme, že je potřebujeme při návštěvě obchodu koupit, není v lidských silách. Řešením může být použití papírového nákupního lístku. Ten lze ovšem velmi lehce ztratit nebo zapomenout doma. Druhým způsobem může být zapisování položek do poznámek v mobilním telefonu. Tento způsob je ovšem značně neefektivní, jelikož je třeba každou položku do písmene vypisovat a není zde možnost jakéhokoliv formátování nebo přídavných funkcí. Ideálním řešením tohoto problému je aplikaci pro mobilní zařízení, která umožní spravovat nákupní lístky a poskytne další funkce, které nám umožní ušetřit čas. Telefon navíc máme vždy při ruce, proto je možné kdykoliv přidávat nové položky. Samotný koncept nákupního lístku v mobilním telefonu není ovšem vůbec inovativní. Vznikla již spousta aplikací, které je možné k tomuto účelu využít. Námi vyvíjená aplikace bude poskytovat ještě další funkce, které ji učiní jedinečnou. Tou hlavní bude inteligentní řazení položek na nákupním lístku podle rozestavení jednotlivých kategorií zboží ve vybraném obchodě. Uživateli by měla umožnit lépe se orientovat na lístku a rychleji obchodem projít. Druhou stěžejní funkcí bude vzdálené přidávání nákupních lístků přes internet. Po registraci na webové stránce bude možné spárovat se s libovolným počtem mobilních zařízení a přidávat na ně vzdáleně nákupní lístky.

Když již máme představu o tom, co bychom chtěli vytvářet, pokusíme se najít co možná nejlepší současné metody pro vedení začínajících projektů, tzv. **startapů**, abychom si zajistili podporu z hlediska metodiky vývoje. Výběrem těchto metod se budeme zabývat v následující sekci.

2.2 Startupy a jejich podpůrné metody

Nejdříve by bylo dobré ujasnit, co že to startup vůbec je. Pokud začneme definicí, dobře ji vystihl ve své knize [15] *Eric Ries*, který označil startup za *"instituci lidí s cílem vytvořit*

nový produkt nebo službu v podmínkách vysoké nejistoty". Zajímavé na této definici je to, co vynechává. Neříká totiž nic o velikosti společnosti, průmyslu nebo ekonomickém sektoru. Nezáleží tedy na tom, zda projekt vzniká ve vládní agentuře, malé firmě nebo nadnárodní korporaci. Zmíněnou vysokou nejistotou je myšleno to, že nevíme, zda naše stěžejní myšlenka, na které by měl být celý startup postaven, je správná a zda uživatelé opravdu náš produkt přijmou a budou ho využívat. Aby byly šance na úspěch projektu zvýšeny, budeme využívat několik metod a principů, které jsou s nimi spojené. Především pak ty, které jsou vyzkoušeny praxí a díky nimž v minulosti již vznikly komerčně úspěšné projekty.

První knihou, ze které budeme čerpat, je **Lean Startup** [15], jejíž autorem je *Eric Ries*. Jeho přístup je založen na častém vytváření prototypů za účelem testování předpokladů o potřebě daného produktu na trhu. Toto testování probíhá za pomoci počátečních zákazníků neboli **early adopters**. Prototypy jsou jim předkládány k otestování a následně je zpracována zpětná vazba, kterou vyjádří svůj dojem z produktu, zejména pak na to, jak velký by o něj měli zájem. Ze zpracovaných výsledků tohoto průzkumu se následně určuje další směr vývoje. Může být rozhodnuto o tom, že je třeba zkusit dělat některé věci trochu jinak. Nebo můžeme dojít k závěru, že se projekt bude ubírat úplně jinou cestou.

Kniha druhá, jejímž autorem je *Ash Maurya*, se jmenuje **Running Lean** [10] a oproti předchozí knize je v popisu metod o něco kokrétnější. Zaměřuje se spíše na webové startupy a poskytuje konkrétní typy a postupy, jak takový startup řídit. Autor v ní tvrdí, že není důležité mít bezchybný plán nebo nápad hned na začátku, ale musíme najít takový, který bude fungovat dříve, než vyčerpáme všechny zdroje. Klíčové principy bychom mohli shrnout jako vytváření pouze takových produktů, o které zákazníci mají zájem, činit správná rozhodnutí ve správný čas, maximálně zrychlit celý proces vývoje a používat techniky, které jsou ověřeny v praxi. My se touto knihou budeme spíše jen inspirovat, než abychom postupy v ní obsažené do detailu dodržovali.

2.3 Early adopters

V předchozích částech byli několikrát zmíněni tzv. early adopters. Pokud bychom je chtěli blíže definovat, můžeme o nich říci, že jsou to uživatelé, kteří aktuálně mají nějaký problém, jehož řešením je náš projekt. Dle [10] musí však early adopter být i tak trochu vizionář, aby byl ochoten zkusit nový produkt a podílet se na jeho vývoji v rámci svého času. Za to mu bude nabídnuta možnost vyzkoušet a používat výsledek dříve než ostatním, což by pro něj mělo být největší odměnou. Může dostat i finanční úlevu z ceny produktu nebo služby, ale obecně by měl být ochoten za vyřešení svého problému zaplatit.

Často také platí, že najít takové uživatele bývá problém. Pro reálný startup je potřeba sehnat takovýchto osob několik desítek, aby zpětná vazba byla co nejkvalitnější. Pro náš projekt se pokusíme sehnat co nejvíce takovýchto uživatelů, kterým by se dalo říkat early adopters, abychom s nimi mohli testovat naše původní nápady a domněnky. Pro další testování využijeme běžné uživatele, kteří mohou také přispět svým názorem na věc a zkvalitnit tak zpětnou vazbu.

2.4 Shrnutí

V této kapitole byly představeny metody vývoje, testování a získávání zpětné vazby, které následně použijeme při realizaci našeho projektu, který bude vznikat v iteracích. Je třeba ovšem říci, že v této práci nebudeme rozebírat jednotlivé iterace, tak jak vznikaly, ale

popíšeme teoretický základ a návrh aplikací, tak jako bychom jejich výslednou formu znali již na začátku a mohli všechny iterace spojit do jediné. Postupné zdokonalování našeho původního nápadu pomocí znalostí získaných ze zpětné vazby od early adopters a dalších zdrojů zmíníme až v kapitole 6.

V následujících třech kapitolách se tedy budeme zabývat pouze teoretickým pozadím a návrhem. Začneme popisem mobilní aplikace.

Kapitola 3

Mobilní aplikace

Pokud jsme deklarovali, že budeme vyvíjet aplikaci pro mobilní zařízení, bude nutno nejprve určit, pro jakou platformu, jaký typ a v jakém programovacím jazyce vlastně chceme tvořit. Ideálním stavem by bylo vybrat jeden jazyk a vytvořit univerzální aplikaci, která by fungovala na všech zařízeních různých platforem a navíc ještě vypadala vždy úplně stejně. Možné to sice je, ale přináší to s sebou jistá úskalí, která budou rozebrána v následující sekci. Na jejich základě se pak rozhodneme, jaký typ a případně pro jakou platformu budeme aplikaci vyvíjet.

3.1 Typy mobilních aplikací

Mobilní aplikace bychom podle [9] mohli rozdělit do dvou kategorií. V první jsou aplikace, které existují ve tvaru binárních souborů a jsou spouštěny přímo na procesoru mobilního zařízení. Říká se jim **nativní aplikace** a jsou vyvíjeny speciálně pro určitou platformu, kde z těch nejznámějších můžeme zmínit *Android, iOS, BlackBerry, HP webOS, Symbian OS* a *Windows Mobile*. Do druhé kategorie pak patří aplikace, které jsou k uživateli doručovány skrze protokol *HTTP* a zobrazovány ve webovém prohlížeči. Z větší části jsou tedy spuštěny na vzdáleném serveru, ale jejich část může být prováděna přímo v zařízení (např. *JavaScript*). Mohli bychom říct, že místo spuštění programu přímo v operačním systému zařízení si vlastně pouze prohlížíme webovou stránku, která se snaží napodobit nativní aplikaci.

Vývoj nativních aplikací je specifický pro každou platformu. Jsou zde různé programovací jazyky, vývojová prostředí, debugovací nástroje a emulátory. Pokud bychom chtěli vytvářet nativní aplikaci pro všechny nejpopulárnější platformy, zabral by vývoj delší dobu, protože by musel probíhat odděleně. Vývoj aplikací z druhé kategorie je o dost pohodlnější. Obecně stačí vytvořit webovou aplikaci a optimalizovat ji pro malé dipleje. Dobrým řešením může být použití nejnovějšího *HTML5* ve spojení s webovým frameworkem *JQuery*, který je určen přímo pro dotyková mobilní zařízení. Rozdíl zobrazení na jednotlivých platformách pak bude záležet pouze na použitém prohlížeči. Nyní sice máme metodu, jak jednoduše vytvářet aplikace napříč platformami, ta ale není dokonalá a existuje řada důvodů, proč ji nepoužít.

Prvním důvodem může být možnost interakce s uživatelem a ostatními aplikacemi, která souvisí s integrací do operačního systému. Nativní aplikace mají v tomto směru velkou výhodu, jelikož mohou využívat více funkcí zařízení, na kterém běží. Pokud vezmeme v úvahu dotykové mobilní zařízení, můžeme využít gest jako je dvojdotek (multitouch) nebo tažení

(swipe), ale třeba i rozpoznávat stisknutí hardwarových tlačítek nebo používat další vybavení jako je GPS nebo kamera. Přístup k těmto funkcím je pro webové aplikace běžící v internetovém prohlížeči velmi problematický a v některých případech nemožný. Existuje zde ovšem možnost, jak daný problém obejít. Stačí vytvořit hybridní aplikaci s nativním základem, do kterého vložíme modul pro zobrazování webové aplikace. Na tomto principu je založeno několik komerčních projektů, jako je například *PhoneGap* nebo *Appcelerator*, které dokážou z jednoho kódu na webové bázi vygenerovat nativní aplikace pro všechny známé platformy. Toto řešení ovšem není z daleka ideální. Přesto, že zmíněné služby proklamují, že je díky nim možné dosáhnout na vnitřní funkce zařízení, jejich možnosti jsou stále omezené a ve výsledku máme stále webovou aplikaci, pouze zabalenou do nativní schránky.

Dalším podstatným důvodem je vzhled a zažité ovládání pro danou platformu. Uživatelé totiž očekávají, že ovládací prvky, na které jsou ve svém operačním systému zvyklí, budou fungovat pro všechny aplikace stejně. Příkladem může být vyvolání menu dlouhým dotykem na položku, kterého ovšem není možné ve webových aplikacích dost dobře dosáhnout. Celkový zážitek z používání a vzhledu pak nebude tak silný, jako u nativní verze. Našli bychom i další důvody, jako třeba problém nalezitelnosti, možnosti monetizace a v neposlední řadě i potřebu internetového připojení ke spuštění webových aplikací.

Závěr je takový, že nativní aplikace jsou stále pro vývoj komplexních aplikací lepší volbou. Jinak řečeno, nativní aplikace vypadají lépe, snadněji se ovládají a jde s nimi vytvořit složitější funkčnost než s webovými aplikacemi spouštěnými přes prohlížeč. K tomu se kloní většina vývojářů a důkazem toho mohou být nejlepší mobilní aplikace současnosti, které jsou vytvářeny právě tímto způsobem. Webové technologie se však rozvíjejí velmi rychle a dá se předpokládat, že v budoucnu se možnosti obou přístupů vyrovnají. V naší práci se ale budeme soustředit na nejnovější trendy a budeme vytvářet aplikaci v nativní podobě. Kvůli náročnosti vývoje a omezeným zdrojům zatím pouze pro jednu platformu, kterou bude Android.

3.2 Operační systém Android

Jak je uvedeno v [11] a [4], Android je známý operační systém primárně určený pro mobilní zařízení jako jsou mobilní telefony, tablety a GPS navigace, ale je ho možné najít předinstalovaný i na některých netboocích. Je vyvíjen konsorciem *Open Handset Alliance*, které je vedené *Googlem* a zastřešuje firmy jako je *Ebay, LG, Samsung, HTC, Intel, T-Mobile* a mnoho dalších. Je založený na jádru Linux 2.6 a je poskytován pod *Apache licencí*. K dispozici je tedy kompletní zdrojový kód celého systému a výrobci zařízení ho mohou použít k libovolným účelům. Systém je navržen tak, aby mohl být použit na všech možných hardwarových řešeních a umožňoval zobrazování na displejích různých velikostí a rozlišení. Vytvořením jedné vhodně optimalizované aplikace tak může vzniknout produkt kompatibilní s více než tisícem různých zařízení. Díky tomu vznikla kolem tohoto operačního systému široká komunita uživatelů a vývojářů. O popularitě Androidu svědčí i stále vzrůstající počet prodaných telefonů s tímto operačním systémem.

Značnou zásluhu na úspěchu androidu má široká nabídka aplikací, které jsou distribuovány přes **Google Play** (bývalý Android Market), což je přehledná oficiální databáze, ve které je možné vybírat z produktů, z nichž je část zcela zdarma, část je omezena reklamou a část je placená. Přístup k marketu a následné stažení a naistalování aplikace je možné provést přímo z telefonu. Nově lze market procházet i pomocí počítače a vybírat ty aplikace, které se následně nainstalují do telefonu, což může být pro uživatele pohodlnější a ušetřit jim čas. Existuje zde i velké množství aplikací realizujících funkci nákupního lísku a

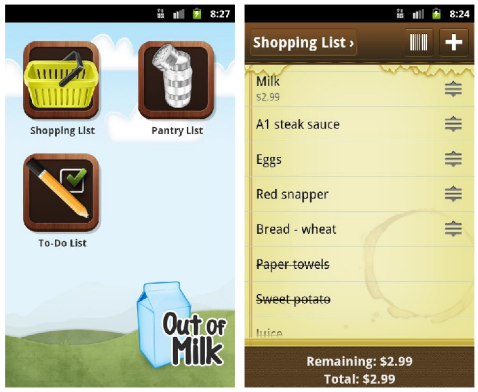
na nejvýznamnější z nich se podíváme v následující sekci.

3.3 Konkurenční aplikace

Počet aplikací pro operační systém Android, které si kladou za cíl nahrazení nákupního lístku, existuje relativně velký počet. Je to zejména z důvodu jednoduchého základního konceptu, který stačí na vytvoření aplikace vhodné k použití pro tento účel. Budou nás zajímat hlavně ty aplikace, které jsou umístěny na Google Play marketu [2], odkud také byly čerpány informace. Jelikož jsou některé více propracované a jiné méně, podíváme se v následující sekci na ty nejpovedenější z nich.

3.3.1 Out of Milk

Out of Milk je nejpopulárnější aplikací v oblasti nákupních lístků. Její instalační soubor si přes Google Play stáhlo 1 až 5 milionů uživatelů (počet stažení není uváděn přesně). Přičemž její placenou verzi si za necelých sto korun zakoupilo 10 000 až 50 000 lidí. Průměrné hodnocení aplikace od uživatelů, které budeme uvažovat pro placenou verzi, je pak 4,6 z 5. Dá se tedy mluvit o velmi úspěšném projektu, který i přináší zisk.



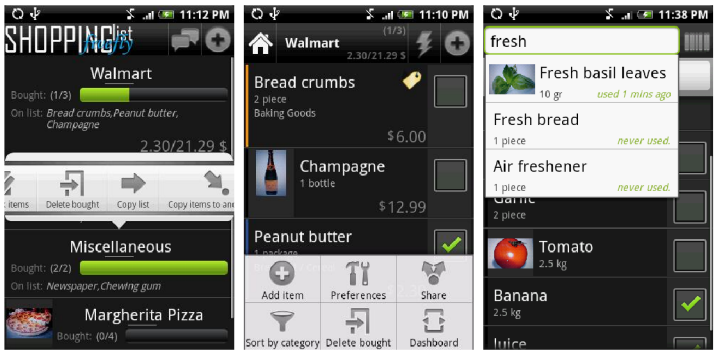
Obrázek 3.1: Obrázky aplikace Out of Milk [2]

Aplikace má propracované uživatelské rozhraní a příjemnou grafiku. Obsahuje opravdu velké množství užitečných funkcí, o čemž svědčí velká uživatelská oblíbenost. Mezi klíčové patří například přidávání položek pomocí hlasu nebo vyfocení čárového kódu a historie nakupovaných položek a jejich cen. V placené verzi je pak možné využít i zálohování dat, propojení s webovou stránkou, na které je možné nákupní lístky upravovat, a nebo synchronizace lístků s jinými uživateli. Mezi další pozitiva můžeme také zařadit fakt, že v základní verzi není reklama.

3.3.2 Fivefly Shopping list

Další z řady úspěšných aplikací pomáhající při nákupu potravin je Fivefly Shopping list, která v sobě nese velké množství funkcí. Její nevýhodou je, že na první dojem může působit jako nepřehledná. To je ostatně vidět i na obrázcích 3.2. Nepřehlednost ještě zvyšuje reklama, která je v neplacené verzi zobrazena v horní části a lze se jí zbavit pouze zakoupením licence za přibližně padesát korun. Na druhou stranu lze použít pokročilé funkce,

jako je sdílení nákupních lístků bez nutnosti cokoliv platit. Aplikace dosáhla průměrného hodnocení 4,5 z 5. Její bezplatnou verzi si stáhlo 500 000 až 1 000 000 lidí a její placenou verzi si zakoupilo 1 000 až 5 000 lidí.

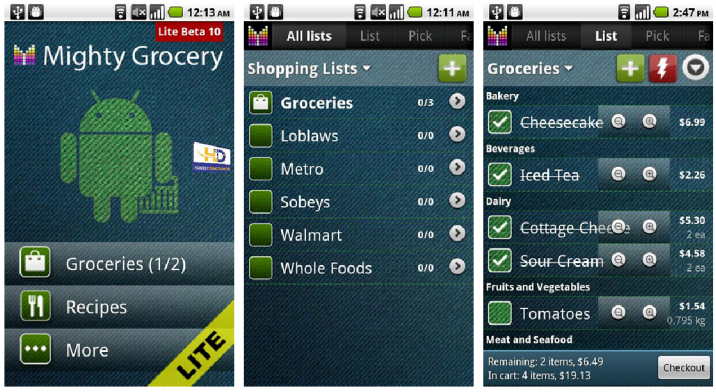


Obrázek 3.2: Obrázky aplikace Fivefly Shopping list v plné verzi (bez reklam)[2]

Aplikace nabízí ale i funkce, které u Out of Milk chyběly. Je to například řazení položek do kategorií včetně jejich editace nebo přidání fotografií k položkám. Naopak chybí spojení s webovým rozhraním a přidávání položek výběrem z minulých nákupů. Za vyzdvihnutí stojí funkce našeptávání při zadávání nové položky, která je schopna poradit i české položky včetně těch, které byly uživatelem zadány dříve.

3.3.3 Mighty Grocery Shopping

Tato aplikace poskytuje jednoznačně největší množství funkcí a možností. Je nabízena ve freemium modelu, kde si neplacenou verzi doposud nainstalovalo 100 000 až 500 000 lidí a placenou 10 000 až 50 000 lidí, při ceně 77 Kč. Tato aplikace tedy není tak úspěšná, jako Out of milk, i když obsahuje více funkcí a má také velmi pěkně zpracované uživatelské rozhraní. Mohlo by to být způsobeno velkou složitostí aplikace. Průměrné hodnocení je pak 4,5 z 5.



Obrázek 3.3: Obrázky aplikace Mighty grocery shopping list [2]

Jak již bylo zmíněno výše, aplikace má opravdu velké množství funkcí, včetně synchronizace dat s webovou aplikací. Velká část z nich je ovšem přístupná pouze v placené verzi.

Čím se ale aplikace liší od ostatních, je velké množství způsobů, kterými můžeme přidat položku na lístek. Lze to ze seznamu kategorií, z oblíbených položek, z historie a dokonce i z receptů. Velká nevýhoda této aplikace ovšem je, že i zkušenému uživateli trvá poměrně dlouho dobu, než se v aplikaci zorientuje. Pro běžného uživatele to pak musí být velký problém a raději sáhne po jednodušší aplikaci, i když přijde o některé funkce, které pro něj nejsou nezbytné. Další nevýhodou této aplikace je pomalá reakce při některých operacích, což je velmi nepříjemné.

3.3.4 Závěrečné zhodnocení konkurence

Jak je vidět konkurence je pestrá a aplikace obsahují všechny možné funkce. Některé jsou více propracované a sofistikované, jiné jsou zase založené na jednoduchém principu. V tabulce 3.1 jsou přehledně zobrazeny největší klady a zápory jednotlivých aplikací. Můžeme zde vidět, že každá z nich má své silné stránky, ale zároveň žádná z nich není dokonalá. Krom toho také z průzkumu vyplynulo, že žádná z aplikací neobsahuje funkci, která by ji propojila s obchodem, ve kterém bude nákup proveden. To bylo ověřeno i mezi aplikacemi, které jsou na marketu a zde nejsou popsány. Tím jsme si ověřili, že naše funkce inteligentního řazení položek je unikátní

Název aplikace	**Plus**	**Mínus**
Out of Milk	Příjemné uživatelské rozhraní, pokročilé funkce	Řazení položek do kategorií
Fivefly Shopping list	Kategorie, obrázky položek, české položky v databázi	Nepřehledné uživatelské rozhraní, žádné webové rozhraní
Mighty Grocery Shopping	Nejvíce funkcí, různé možnosti přidání položky na lístek	Složitost, místy pomalejší

Tabulka 3.1: Shrnutí kladů a záporů konkurenčních aplikací

3.4 Vývojové nástroje a prostředí

Abychom mohli začít vyvíjet aplikace pro Android, budeme potřebovat jisté softwarové nástroje, které nám umožní zkompilovat kód v Javě a následně ho převést do podoby spustitelného souboru operačního systému Android. Těmi jsou **JDK (Java Development Kit)** a **Android SDK (Software Developtment Kit)**. Pokud si chceme usnadnit práci a nepřekládat celý projekt v příkazové řádce, nainstalujeme jeden z IDE editorů. Oficiálně podporovaný je editor Eclipse, pro který byl dokonce vydán plugin **ADT (Android Developtment Tools)** pro práci s Android projekty.

Součástí Android SDK je také emulátor operačního systému Android, který dovoluje spouštět aplikace na virtuálním zařízení s požadovanou verzí systému a parametrů a umožňuje tak testovat bez nutnosti použití zařízení fyzického. Z parametrů je možné si nastavit emulované hodnoty, jako je například velikost paměťové karty a typ připojení. Možné je také simulovat aktuální GPS pozici. Hlavní předností ovšem je možnost nastavení velikosti a rozlišení displeje. Díky tomu můžeme zjistit, jak bude naše aplikace vypadat na různých zařízeních od nejmenšího mobilního telefonu až po tablet. Emulátor ale má i

své omezení a některé funkce na něm testovat nelze. Je to například fyzické natočení nebo příjem telefonního hovoru. Z tohoto důvodu je dobré mít na testování alespoň jedno reálné zařízení. Pokud na něm navíc zapneme režim ladění, vznikne po připojení k počítači možnost nahrávat nové verze aplikace přímo z editoru a využívat funkce debugování nebo čtení logu, stejně jako u emulátoru.

3.5 Struktura Android projektu

Stejně jako v jiných Java projektech, i zde je překladový systém upořádán kolem specifické stromové struktury adresářů, která je ale v jistých aspektech jedinečná. V následující sekci bude tato struktura krátce popsána. Začneme s obsahem kořenového adresáře projektu, který je popsán v [12]:

- `assets/`: Složka se statickými položkami, které budou přibaleny k aplikaci.
- `bin/`: Po kompilaci je zde uložen spustitelný soubor aplikace.
- `gen/`: Umístění kódu vygenerovaného při překladu aplikace
- `libs/`: Složka s Java knihovnami třetích stran, které jsou v aplikaci použity.
- `src/`: Zdrojové kódy aplikace v jazyku Java.
- `res/`: Složka, která uchovává prostředky, jako jsou ikony nebo grafické návrhy uživatelského rozhraní, ale třeba také soubor s textovými řetězci.
- `AndroidManifest.xml`: Je soubor ve formátu XML, který deklaruje obsah aplikace a tvoří tak její základ. Obsahuje informace o jejích aktivitách, službách, ale třeba i o tom, jak budou tyto komponenty propojeny s operačním systémem. Například je zde uvedeno, jaká aktivita by měla být vyvolána po spuštění aplikace a současně tak umístěna v hlavním menu zařízení. Dále jsou zde například deklarována práva, určující k jaký zdrojům může aplikace přistupovat, a mnoho dalšího.
- `build.xml`: Soubor, kde je uveden skript pro kompilaci aplikace.

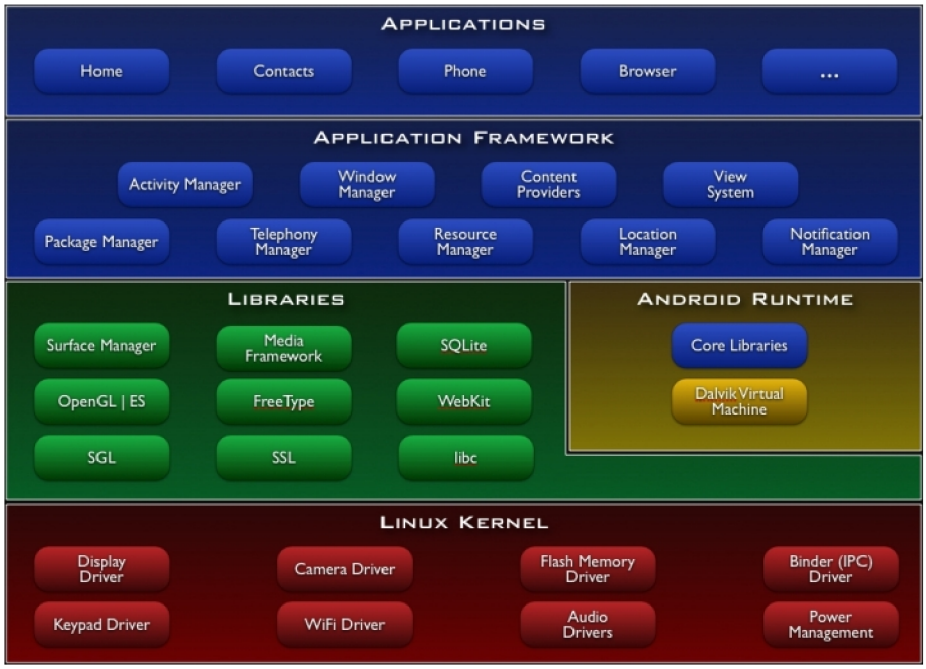
Po každé kompilaci projektu se nám ve složce `gen/` vytvoří důležitý soubor `R.java`, který obsahuje konstanty vytvářející zkrácené odkazy na zdroje, které jsme umístili do stromové struktury našeho projektu. Díky tomu se nám do ruky dostává silný nástroj, jak můžeme v našem zdrojovém kódu přímo přistupovat k jednotlivým prostředkům našeho projektu, jako jsou grafické návrhy, ikony nebo například textové řetězce. Pokud tento soubor ještě využijeme ve spojení s IDE editory jako je Eclipse, které obsahují funkci automatického dokončování kódu, přístup ke zdrojům se stane ještě rychlejším.

V souvislosti se strukturou projektu je také třeba zmínit funkci **kvalifikátorů** Android aplikací. Ty dokážou vybírat jeden typ zdroje, například soubor s textovými řetězci, z několika různých složek v závislosti na nastavení nebo vlastnostech zařízení, na kterém jsou spouštěny. Díky tomu můžeme velice jednoduše aplikace vytvářet v několika jazycích nebo je přizpůsobovat různým velikostem displejů cílových zařízení. Abychom složku specifikovali pro nějakou vlastnost, umístíme na konec jejího názvu řetězec `-<požadovaná_vlastnost>`. Pokud chceme tedy mít aplikaci ve dvou jazykových mutacích, jednoduše vytvoříme složky `res/values-cs` a `res/values-en` a do každé z nich umístíme soubor se stejným jménem `string.xml`, který bude obsahovat řetězce v příslušném jazyce. Výběr správné složky pak

probíhá automaticky a vývojář ji nemůže nijak ovlivnit. Pokud ale není nalezena složka příslušného nastavení, jsou zdroje čerpány ze složky základní. Pro náš příklad by se v případě jiného než anglického nebo českého prostředí brala data ze složky `res/values`. Těchto zkratek, které se používají k přizpůsobení aplikace pro konkrétní prostředí, je velké množství. Pro náš projekt je budeme využívat zejména k optimalizaci grafického rozhraní, která je popsána v sekci 3.14.

3.6 Architektura operačního systému

Abychom mohli vyvíjet Android aplikace, je třeba znát alespoň základní architekturu. Ta je pro tento operační systém rozdělena dle [4] do pěti hlavních vrstev, viz obrázek 3.4:



Obrázek 3.4: Architektura operačního systému Android [4]

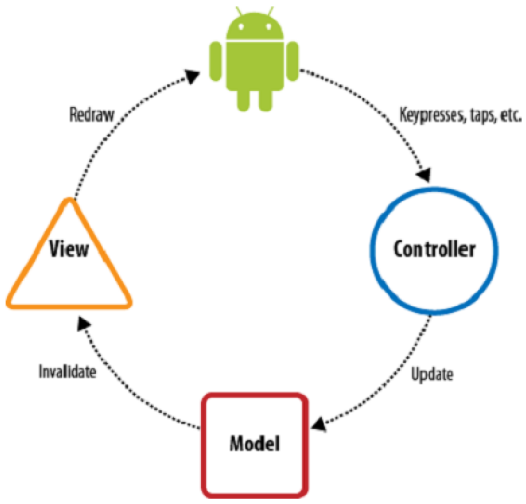
- **Linux kernel**: Linuxové jádro operačního systému, které tvoří abstraktní vrstvu mezi používaným hardwarem a zbytkem softwaru ve vyšších vrstvách. Pro nás nebude tato vrstva tolik zajímavá, protože do ní nebudeme zasahovat. Využijeme pouze jejích metod, které vrací vstupní data od hardwarových prvků, jako je dotykový displej nebo tlačítka.
- **Android Runtime**: Vrstva starající se o spuštěné procesy. Každá aplikace běží ve svém vlastním procesovém vlákně spolu s vlastní instancí virtuálního stroje *Dalvik*. Abychom ze zdrojového kódu v Javě získali spustitelný soubor kompatibilní s *DVM (Dalvik virtual machine)* musíme nejprve celý projekt zkompilovat do Java byte kódu, a to pomocí stejného kompilátoru, který používáme v případě překladu běžných Java aplikací. Následně je překompilován Java byte kód pomocí Dalvik kompilátoru do výsledného Dalvik byte kódu, který již může být spuštěn na DVM. Runtime vrstva je

silně závislá na Linux kernel kvůli funkcionalitě, jako je správa procesů a paměti. My tuto vrstvu využijeme hlavně při spouštění nových procesů na pozadí naší aplikace.

- **Libraries**: Android obsahuje množinu C/C++ knihoven, které jsou používány mnohými komponentami systému. Tyto funkce jsou vývojářům k dispozici pomocí aplikačního frameworku. Mezi knihovny, které využijeme my, patří například SQLite (odlehčená verze databázového systému).
- **Application Framework**: Poskytuje vývojářům přístup k velkému množství služeb a funkcí, které mohou být použity v aplikacích. Dává jim možnost přistupovat k hardwaru, spouštět služby na pozadí nebo nastavovat upozornění. Pro nás bude rozhraním, přes které budeme vyvolávat služby ostatních vrstev.
- **Applications**: Nejvyšší vrstva, na které běží standardní aplikace, které jsou buď distribuovány spolu se systémem nebo mohou být doinstalovány dodatečně.

3.7 Architektura frameworku uživatelského rozhraní

Po seznámení s architekturou operačního systému využijeme informace z [11] a podíváme se, jak to funguje v samotné aplikaci. Konkrétněji ve frameworku uživatelského rozhraní. Ten je v systému Android, stejně jako jsou jiné Java frameworky, organizován okolo architektury **Model-view-controller (MVC)** zobrazené na obrázku 3.5. Ta rozděluje aplikaci do tří oddělených komponent tak, že modifikace jedné má jen minimální vliv na ostatní.



Obrázek 3.5: Model-View-Controller architektura [11]

Hlavní částí je **model**. Ten obsahuje veškerou vnitřní funkčnost aplikace neboli aplikační logiku. Hlavně ale spravuje datový model, který definuje stav aplikace a realizuje propojení s databází. Více o problému perzistentních dat a jejich uložení bude napsáno v sekci 3.9. Model je nejdůležitější a velmi obsáhlá část aplikace, které se bude věnovat většina sekcí této kapitoly. Data, která model udržuje, se k uživateli dostanou pomocí částí **view**, která se stará o vizualizaci. Pokud dojde ke změně modelu, je této části předána informace,

která část displeje má být překreslena. Jak probíhá samotné vykreslování plochy displeje bude podrobně popsáno v sekci 3.13. Poslední část **controller** se stará o reakce aplikace na vstupní akce, jako je například stisknutí tlačítka, dotek na displej nebo příchozí telefonní hovor. Ty jsou ukládány do vstupní fronty, odkud je framework postupně vyjímá a zpracovává. Komunikuje přitom s modelem, který na základě těchto informací aktualizuje datový model a stav aplikace. V případě potřeby pak pomocí části view vyvolá překreslení displeje.

3.8 Komponenty aplikace

Komponenty jsou speciální programové struktury, které zajišťují stěžejní funkce aplikace, jako je přístupový bod pro spuštění, přechod mezi okny nebo volání služeb. Každá komponenta tvoří speciální entitu, která pomáhá definovat všeobecné chování aplikace. Při vývoji se použití těchto struktur nevyhneme. V systému Android existují celkem čtyři základní druhy komponent, přičemž každý druh obsahuje specifickou funkci a životní cyklus, který definuje, jak komponenta vzniká a jak zaniká. Tyto čtyři typy jsou dle [1]:

- **Aktivity**: Jsou základní obrazovky s uživatelským rozhraním. Aplikace obvykle mívají více aktivit, mezi kterými může uživatel přecházet. Aktivity mohou být spuštěny i jinou aplikací, než pro kterou jsou vytvořeny a posílat si mezi sebou data. Například může být v aplikaci vyvolána aktivita fotoaparátu, která pořídí fotku. K vyvolání tohoto přechodu slouží tzv. intenty, o nichž se zmíníme na konci této sekce.

- **Služby**: Služba je komponenta, která běží dlouhodobě na pozadí a neposkytuje grafické uživatelské rozhraní. Typickým příkladem služby je aplikace, která běží na pozadí a přehrává hudební soubory, zatím co uživatel může pracovat s jinými aplikacemi. Služby mohou být spuštěny jinou komponentou, jako je například aktivita.

- **Poskytovatelé obsahu**: Spravují a sdílejí aplikační data, která mohou být uložena v souborovém systému, SQLite databázi, webu nebo jakémkoliv jiném úložišti perzistentních dat, ke kterému má aplikace přístup. Pomocí poskytovatele obsahu se mohou ostatní aplikace dotazovat nebo dokonce měnit cizí aplikační data, pokud je to povoleno. Jako příklad můžeme uvést přístup k úložišti kontaktů, které je právě spravováno jedním poskytovatelem obsahu a umožňuje tak modifikovat nebo přidat kontakt z jakékoliv aplikace. Mohli bychom tedy říci, že se jedná o jakési datové rozhraní.

- **Příjemci broadcastu**: Jsou komponenty, které odpovídají za broadcastové oznámení, díky kterému je možné poslat zprávu všem aplikacím v systému, jako je například upozornění na nízký stav baterie. Toto oznámení mohou vyvolat i samy aplikace. Například ostatním aplikacím dát vědět, že do zařízení byla stažena data a nyní jsou již pro všechny dostupná. Příjemci broadcastu ovšem nemají grafické uživatelské rozhraní, pouze mohou zobrazit upozornění do stavového řádku a dát tak vědět uživateli, že byla broadcoastová událost vytvořena.

Tři ze čtyř výše zmíněných základních komponent – *activities, services* a *broadcast receivers* – jsou aktivované asynchronními zprávami, které se nazývají **intenty** [11]. Ty slouží k propojení jednotlivých komponent za běhu aplikace. Můžeme na ně pohlížet jako na akce, které se mají vykonat, a to nejen v rámci naší aplikace, ale globálně. Pomocí intentu můžeme například vyvolat akce, které vedou k otevření webové stránky nebo telefonního seznamu.

Důležité přitom je, že k intentu můžeme připojit data, která budou předána cílové komponentě. Zároveň ale také můžeme očekávat, že vyvolaná komponenta vrátí nějaká data nazpět. Můžeme tedy například u zmíněného otevření telefonního seznamu vybrat kontakt, který bude vrácen do naší aplikace. Na druhou stranu také můžeme vytvořit itentem požadavek na akci, pro kterou dopředu nevíme, kterou aplikací bude zpracována. Pokud například v systému existuje více možností, jak odeslat zprávu (např. sms, Icq nebo Facebook Messenger), systém uživateli nabídne, aby si zvolil, jak se má daný intent zpracovat.

3.9 Uložení perzistentních dat

Platforma Android poskytuje několik možností, jak uložit persistentní data. Výběr jednotlivých eventualit závisí převážně na potřebách a požadavcích na vyvíjenou aplikaci. Některé metody uložení mohou být oproti ostatním pomalejší, neposkytovat možnost sdílet data s ostatními aplikacemi nebo například nezvládají uložit velké objemy. V naší aplikaci budeme využívat více možností uložení pro různé typy dat, která bude potřeba trvale uchovat. Metody uložení jsou dle [3] následující:

- **Shared Preferences**: Třída *SharedPreferences* poskytuje řešení pro perzistentní uložení párů klíč a hodnota, kde hodnota může být různých primitivních datových typů. Data uložená pomocí této metody přetrvají mezi jednotlivými spuštěními aplikace i přes její násilné ukončení. Data se neztratí ani při aktualizaci. Odstraněny jsou až ve chvíli, kdy uživatel aplikaci odinstaluje nebo v systému provede manuální vymazání aplikačních dat.

- **Interní úložiště**: Uložení dat přímo do privátního prostoru interní paměti zařízení v podobě souborů. Za normálních okolností je takto uložený soubor privátní pro aplikaci, která ho vytvořila. Je však možné nastavit práva tak, aby k němu měl přístup uživatel i ostatní aplikace, a to se všemi riziky editace a smazání.

- **Externí úložiště**: Každé Android-kompatibilní zařízení podporuje uložení souborů do externího úložiště. Tím může být například vyjímatelná paměťová karta nebo paměť vlastního zařízení. Soubory uložené do externího úložiště jsou globálně přístupné ke čtení a mohou být i modifikovány uživatelem při zapnutí přenosu dat do počítače (USB mass storage).

- **Databáze**: Android poskytuje plnou podporu *SQLite* databází, které jsou po vytvoření dostupné všem třídám a to pouze v dané aplikaci. Jelikož je budeme hojně využívat k ukládání velkého množství dat, bude tento typ databáze popsán v následující sekci 3.10.

- **Připojení k síti**: Poslední možností je přistupovat k datům pomocí vlastní webové služby, která je realizována připojením zařízení k síti. Můžeme tak například přijímat a odesílat data do *MySQL* databáze, která je napojena na webové rozhraní. Toho ostatně budeme využívat i my v naší aplikaci. Návrhem a rozborem webové aplikace a komunikací s ní se zabývá kapitola 4.

3.10 SQLite databáze

SQLite je velmi oblíbená databáze, kterou dodává ve svých komerčních zařízeních a projektech velké množství společností (například Apple, Google nebo Symbian), ale je i

součástí hodně open-source projektů (například Mozilla, PHP nebo Python). Jedná se o vestavěnou relační databázi, která neběží jako oddělený proces, ale je součástí procesového prostoru aplikace. Všechny databázové tabulky, trigery, pohledy a další jsou ukládány přímo na disk do jednoho souboru. Díky tomu a svému SQL rozhraní zanechává tato databáze velmi malou paměťovou stopu a je poměrně rychlá. Krom toho je práce s ní pro vývojáře, kteří již mají zkušenosti s SQL databázemi, jednoduchá a intuitivní.

Zajímavou vlastností, která se týká SQLite databází na platformě Android, je možnost provést tzv. **upgrade**. Pokud vytváříme aplikaci, je pravděpodobné, že postupem času, jak budeme přidávat nové funkce a rozšíření, bude třeba i měnit strukturu databáze. Abychom byli schopni odlišit jednotlivé verze, existuje zde možnost označit verzi databáze přirozeným číslem, počínaje jedničkou. Pokud aplikace zjistí, že číslo aktuální verze databáze v zařízení je nižší než verze aktuálně spouštěná, vyvolá metodu *onUpgrade*, kam můžeme umístit kód zajišťující změny databáze tak, aby odpovídala té aktuální. Pokud přidáváme sloupce do tabulky, jde o triviální operaci pomocí SQL příkazu ALTER TABLE. Ten je zde ovšem omezen a kromě přidávání sloupců nepodporuje další operace. Pokud tedy chceme provádět složitější změny, musíme vytvořit novou tabulku a data do ní, i s případnými modifikacemi, překopírovat.

3.11 SQLite FTS3 rozšíření

FTS3 (Full-text search) je rozšíření SQLite databáze, které přináší speciální druh tabulky umožňující fulltextové vyhledávání v uložených hodnotách. Je možné vyhledat řádky obsahující hledané slovo uvnitř uloženého řetězce nebo díky použití základních regulárních výrazů vyhledávat pouze část slova. Obyčejné tabulky na proti tomu umožňují pouze vyhledávat přesné shody hledaného řetězce s uloženým. Tabulku v FTS3 rozšíření nazýváme **virtuální**, protože ve skutečnosti se skládá ze tří až pěti obyčejných, tzv. *stínových tabulek*. Ty obsahují naindexovaná data sloužící ke zrychlení a vylepšení textového vyhledávání. Z toho vyplývá, že virtuální tabulka sice zabírá více místa na disku a její vytváření trvá o něco déle, ale vyhledávání textových řetězců v ní je značně rychlejší oproti obyčejné tabulce. Hlavně ale umožňuje fulltextové vyhledávání, které běžné tabulky nepodporují, a které budeme využívat i v naší aplikaci. Pozor si přitom musíme dát na to, že ve virtuální tabulce není možné implicitně určit primární klíč. Pokud tak budeme potřebovat identifikovat jednotlivé položky, musíme si sloupec s jednoznačnými idenfikátory vytvořit a spravovat sami.

Virtuální tabulky obsahují navíc několik příkazů, které implementují právě ony speciální rozšiřující funkce. Pro náš projekt bude nejdůležitějším operand **MATCH**, který implementuje fulltextové vyhledávání. Je součástí *WHERE* klausule ve výběrovém příkazu *SELECT*. Zde je několik základních příkladů použití:

1. *SELECT * FROM table WHERE docs MATCH 'linux';*
2. *SELECT * FROM table WHERE docs MATCH 'linux application';*
3. *SELECT * FROM table WHERE docs MATCH 'lin*';*

V první případě jsou v databázi vyhledány všechny řádky, které někde v řetězci, uloženém ve sloupci *docs*, obsahují slovo *linux*. Druhý příkaz vybere řádky, které někde v daném sloupci obsahují zároveň slova *linux* a *application*. V případě, že bychom mezi oba řetězce umístili klíčové slovo *OR*, byly by vyhledány všechny řádky, které v daném sloupci obsahují slovo *linux* nebo *application*. Poslední příkaz pak nalezne všechny řádky, které obsahují

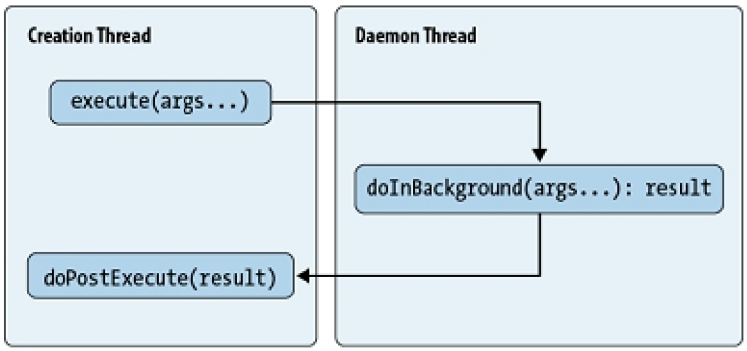
slovo začínající na *lin*.

V předchozích dvou sekcích o SQLite bylo čerpáno z publikace [8].

3.12 Práce s vlákny

Jak uvádí [11], základním předpokladem dobré aplikace je její plynulé fungování bez zamrzání. To znamená povinnost zareagovat na jakýkoliv uživatelský vstup během 200ms. V případě, že aplikace nebude schopna odpovědět ani do 5 sekund, aktivita bude *ActivityManagerem* ukončena jako neodpovídající. A to i případě, že nedošlo k chybě a stále v ní probíhá kontrolovaný výpočet. Jádro tohoto problému je v tom, že grafické uživatelské rozhraní v Androidu, stejně jako ve většině ostatních podobných systémů, běží v jednom samostatném vlákně. To musí pružně odpovídat na uživatelské požadavky a rychle překreslovat displej. Pokud ho zatížíme dalším výpočtem, jakým může být například zpracování dat, nejen že dojde ke zpomalení odezvy aplikace, ale může dojít i k zmiňovanému násilnému ukončení aplikace s hláškou, že aktivita nereaguje.

Tomuto problému se můžeme vyhnout použitím konkurentního přístupu. Z hlavního vlákna uživatelského rozhraní je totiž možné díky třídě **AsyncTask** vytvořit vlákno nové, které poběží v pozadí. Jeho životní cyklus je tvořen několika klíčovými metodami této třídy. První z nich je *execute*, která spouští samotný proces vytváření nového vlákna. Ještě před jeho vytvořením je ve vlákně uživatelského rozhraní zavolána metoda *onPreExecute*, která může obsahovat inicializaci. Není ovšem povinností ji implementovat, pokud jí nebude třeba. Následně třída AsyncTask vytvoří nové vlákno, ve kterém je konkurentně spuštěn kód umístěný v *doInBackground*. Jakmile je dokončen, vlákno je automaticky vymazáno ze systému a je spuštěno *onPostExecute*, a to opět v uživatelském rozhraní.



Obrázek 3.6: Předání dat mezi vlákny [11]

Tento životní cyklus byl vytvořen zejména kvůli možnosti předávání dat mezi oběma vlákny, které je graficky znázorněno na obrázku 3.6. Původní vlákno, vyobrazené vlevo, předává do doInBackground data pomocí parametru spouštěcí metody execute, který je typu *Varargs*. To nám dovoluje uložit proměnné množství parametrů jednoho datového typu a tedy předat tak libovolné množství hodnot. Poté, co nové vlákno ukončí svou činnost v pozadí, je předán výsledek jeho výpočtu, který může být libovolného datového typu, jako parametr do metody onPostExecute v původním vlákně. Takto proběhne předání dat zpracovaných v pozadí do aktivního vlákna uživatelského rozhraní, kde je možné je dále využít.

My budeme vícevláknového přístupu využívat zejména při komunikaci s webovým serverem. Při využití pouze jednoho vlákna by bylo rozhraní naší aplikace po dobu přenosu dat blokováno, což by mohlo při delší odezvě serveru nebo přenosu většího množství dat způsobit zamrznutí nebo pád aplikace.

3.13 Systém vykreslování

Každá plocha displeje je z pohledu zobrazení složena ze stromu komponent, jež jsou potomky třídy *android.view.View*, a nazývají se **pohledy**. Ty, které jsou ve stromu na pozici listu, se nazývají *widgety* a provádějí největší část vlastního vykreslování. Vnitřní uzly, nazývané *kontejnery pohledů*, mohou obsahovat jiné komponenty jako své potomky a nestarají se o vykreslování, ale zajišťují umístění a správné zarovnání pohledů svých potomků na plochu displeje.

Vykreslovací proces začíná v kořenu a prochází postupně celým stromem. Každý kontejner je přitom zodpovědný za vyvolání vykreslovací funkce každého svých potomků a každý pohled je zodpovědný za vykreslení sama sebe. Jelikož je strom procházen postupně systémem *preorder*, rodič je vždy vykreslen dříve než potomek. Pohledy, které jsou na pozicích listů, jsou díky tomu vykresleny v popředí. Vykreslovací proces je ale ve skutečnosti o něco složitější. Aby se dosáhlo větší efektivity, překresluje se pouze v případě, že došlo k zneplatnění nějaké části daného pohledu. Pokud s pohledem není manipulováno, bylo by zbytečné ho překreslovat. Stejně tak přeskočíme vykreslování u rodiče, u kterého s jistotou víme, že bude v budoucnu překreslen neprůhledným potomkem.

Strukturu grafického rozhraní je možné v aplikacích definovat dvěma způsoby. Buďto dynamicky přímo v Java kódu, nebo staticky pomocí XML souboru. Ačkoliv by bylo technicky možné definovat veškeré prvky rozhraní dynamicky, používá se tato metoda pouze za běhu programu na úpravy, které při statické definici ještě nemohou být provedeny. Není tedy problém oba způsoby kombinovat. Převážná většina z uživatelského rozhraní je ale vytvořena staticky v XML souborech, protože to umožňuje oddělit část aplikace, která se stará o zobrazování, od kódu, který definuje její chování. To znamená, že můžeme modifikovat vzhled aplikace, aniž bychom museli zasahovat do kódu a znovu ho kompilovat. To nám umožňuje mít v jednom instalačním souboru více XML souborů s definicemi vzhledů pro různá zařízení.

O samotné vykreslování se stará jádro systému a my ho tedy nemusíme implementovat. Důležité je ovšem postupné vykreslování pohledů ve stromové struktuře, ze kterého budeme vycházet, až budeme sestavovat grafickou podobu naší aplikaci. Stejně tak je podstatné znát rozdíl mezi statickým a dynamickým definováním grafického rozhraní. S vykreslováním také souvisí optimalizace grafického rozhraní pro různá zařízení. Na tento problém se podíváme v následující sekci.

3.14 Optimalizace grafického rozhraní pro různá zařízení

Jak již bylo řečeno, operační systém Android je nainstalován na velkém množství zařízení. To s sebou ovšem přináší komplikace v podobě kompatibility aplikací, které musí být přizpůsobeny mnoha různým displejům. Každý z nich má totiž svou specifickou fyzickou velikost a rozlišení. Android naštěstí poskytuje prostředky, kterými lze dosáhnout toho, že bude aplikace vypadat na všech displejích úplně stejně.

Nejprve je třeba vyhnout se používání pixelů jako měrné jednotky. Pokud bychom tak

nečinili, jeden prvek grafického rozhraní bude na každém displeji jinak velký. Každý pixel můžeme mít totiž jinou fyzickou velikost na různých displejích. Pokud bychom měli dva stejně velké displeje s různými rozlišeními, bude mít zařízení s menším rozlišením menší **hustotu** pixelů a tedy i fyzicky větší pixely. Prvek, jehož velikost bude uvedena v pixelech, se na něm pak bude jevit fyzicky větší, než na displeji s větší hustotou. Abychom tomuto zabránili, měli bychom si osvojit používání na *hustotě nezávislých pixelů*, značených jako **dp**. Jejich základní velikost je odvozena od displeje s hustotou 160 dpi, na kterém se bude jeden dp rovnat jednomu reálnému pixelu. Pokud bude rozlišení displeje jiné, systém dopočítá reálnou velikost prvku v pixelech ze vztahu $px = dp * (dpi/160)$. To zaručí, že bude prvek na každém displeji fyzicky stejně velký.

Druhou věcí, kterou bychom měli dodržovat, je poskytování obrázků, které budou zobrazeny v aplikaci, v několika rozlišeních. Systém sice dokáže v případě potřeby jednotlivé obrázky uložené v jednom rozlišení automaticky přizpůsobit displeji, to se ale negativně projeví na jejich kvalitě. Proto je vhodné pro různé hustoty displeje poskytnou obrázky v různých rozlišeních, aby nemuselo docházet ke zbytečně velkým změnám velikostí. Zde využijeme kvalifikátorů, jež byly popsány v sekci 3.5. Displeje rozdělujeme podle hustoty do čtyř skupin: *ldpi* (malá), *mdpi* (střední), hdpi (vysoká) a *xhdpi* (extra vysoká).

Třetím základním prvkem, který můžeme využít, je definice různých rozhraní pro různé velikosti displejů. Android sice dokáže automaticky přizpůsobit rozhraní pro jednotlivé displeje, ale abychom mohli efektivně využít extra prostoru na velmi velkém displeji, musíme speciálně pro něj definovat vlastní rozhraní. To bude mít jinak definováno rozložení prvků, aby bylo místo lépe využito. Toho docílíme opět pomocí kvalifikátorů. Displeje rozdělujeme podle velikosti do čtyř skupin: *small,normal,large* a *xlarge*.

Posledním základním prvkem, kterým můžeme podpořit správné vykreslování na různých zařízeních, je explicitní deklarace velikostí displejů v souboru *manifest.xml*, které naše aplikace podporuje. Tím zabráníme tomu, aby si ji uživatelé, jejichž zařízení není podporováno, mohli stáhnout a nainstalovat. Toto omezení je účinné při distribuci aplikace přes Google Play market.

Pokud to shrneme, vývoj pro operační systém Android má sice tu nevýhodu, že aplikace musí být optimalizována pro zobrazení na velké množině zařízení, na druhou stranu jsou ale poskytnuty takové nástroje, jenž nám umožňují pohodlně tuto optimalizaci provést.

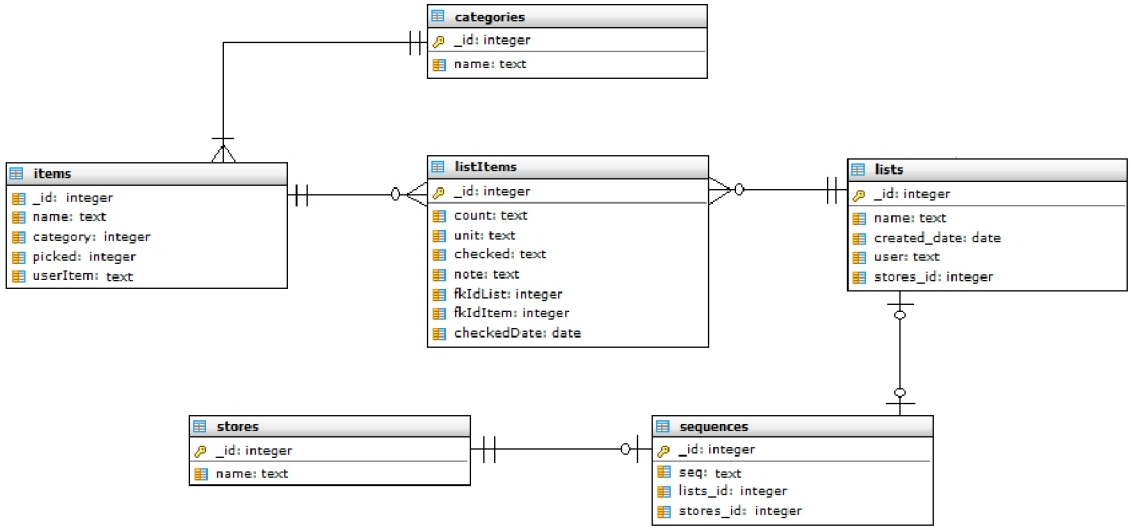
Ve dvou předchozích sekcích bylo čerpáno z [11].

3.15 Návrh datové struktury

Nyní, když již známe teoretické pozadí pro vytvoření aplikace v operačním systému Android, můžeme se pustit do navrhování. A vhledem k tomu, že budeme potřebovat v aplikaci ukládat větší množství dat různých datových typů, podíváme se nejprve na způsob jejich ukládání. Jako prostředek byla zvolena databáze SQLite, popsaná v sekci 3.10. Její struktura je vymodelována na obrázku 3.7. Na něm můžeme vidět tři hlavní tabulky *lists, listItems* a *items*, které zastřešují ukládání dat pro vytváření lístků a položek. Záměrně zde byla zvolena koncepce oddělené tabulky položek (*items*) od tabulky položek na lísku (*listItems*), aby bylo možné dosáhnout znovupoužitelnosti položek. To nám umožní pamatovat si permanentně kategorii nebo to, kolikrát byla která položka použita na nákupním lístku, i když dojte k jejímu smazání. Za tím účelem byl vytvořen speciální řádek *picked* v tabulce *items*, který slouží jako čítač. Počet by samozřejmě bylo možné získat i ze součtu výskytů položky v tabulce *listItems*. Docházelo by pak ale ke zkreslení dat při vymazání

nepotřebného lístku z aplikace. Více k tomuto problému je uvedeno v sekci 6.3.

Funkčnost aplikace ovšem nebude spočívat pouze v ukládání názvů lístků a položek na nich. Proto budou tabulky obsahovat další atributy, díky nimž bude možné implementovat různá rozšíření. Například budeme evidovat u každé položky požadované množství, jednotku a poznámku. V momentě, kdy budeme procházet obchodem, využijeme atributu *checked*, který označujeme položku jako zakoupenou. Vizuální stránka této funkce bude vidět na obrázcích v sekci 3.16, která bude popisovat návrh vzhledu aplikace. U lístků pak evidujeme datum vytvoření a případně ještě uživatele. Ten má speciální význam pro propojení s webovou aplikací. Pokud bude mít tento atribut nastavenou hodnotu, znamená to, že lístek nebyl vytvořen přímo v zařízení, a jeho hodnota bude udávat jméno autora. Toho bude využito při propojení s webovou aplikací, která je popsána v následující kapitole.



Obrázek 3.7: Schéma databáze pro mobilní zařízení v *Crow's foot* notaci

Další dvě tabulky *stores* a *sequences* nám budou sloužit k ukládání dat pro inteligentní řazení položek, které je podrobně popsáno v kapitole 5. První z nich slouží pro ukládání jednotlivých obchodů, které si bude vytvářet sám uživatel a následně si z nich bude moci vybrat, ve kterém nákup provede. Druhá tabulka bude uchovávat sekvence kategorií. Jak bude uživatel procházet obchodem a postupně zaškrtávat jednotlivé položky na lístku, aplikace si bude pamatovat pořadí příslušných kategorií, které následně překóduje na znaky a v podobě řetězce uloží do tabulky. Každému nákupnímu lístku bude náležet právě jedna sekvence, která bude značit v jakém pořadí je uživatel při nákupu zaškrtal a tedy jaké je rozložení kategorií v příslušném obchodě. Pro případ, že by byl lístek smazán, nebo naopak, že by uživatel při zapnutém inteligentním řazení nevybral obchod, ve kterém provedl nákup, je sekvence přiřazena jak k lístku (*lists*), tak ještě k obchodu (*stores*), přičemž nebude vynucováno přiřazení ani k jednomu z nich.

Na obrázku 3.7 také můžeme vidět jednu virtuální tabulku, kterou je *items*. Jak již bylo řečeno v sekci 3.11, taková tabulka nemá primární klíč. O vytvoření a správu sloupce, který bude obsahovat unikátní číslo a identifikovat tak položky, se budeme tedy muset postarat sami. Tato tabulka byla zvolena jako virtuální, aby bylo možné v jejích řádcích vyhledávat fulltextově. Toho však budeme moci plně využít až v okamžiku, kdy se nám podaří realizo-

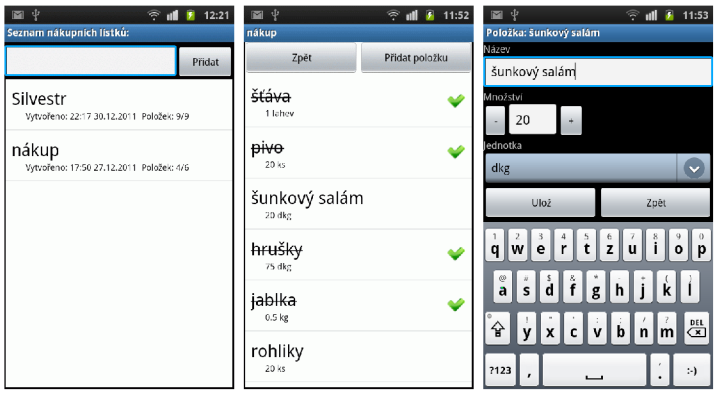
vat další návrh, který předpokládá předvytvoření databáze položek, ze kterých by uživatel mohl vybírat. Idea je zde taková, že uživateli urychlíme zadávání nových položek pomocí našeptávače, který bude nabízet možné dokončení slov. Jako zdroj se použije předem připravená databáze zboží a vlastní položky zadané v minulosti uživatelem.

3.16 Návrh vzhledu aplikace

Před samotnou implementací je dobré si vždy udělat návrh a ujasnit si tak, jak by mělo vypadat uživatelské rozhraní. Zejména pak určit rozmístění jednotlivých funkčních prvků, jako jsou tlačítka. Za tímto účelem vznikla jednoduchá aplikace, jež je zobrazena na obrázku 3.8. Použity byly pouze základní prvky bez jakékoliv optimalizace nebo přídavných funkcí.

Jak již bylo uvedeno, aplikace bude simulovat funkci nákupního lístku. Hned první obrazovka neboli aktivita, proto představuje seznam všech nákupních lístků. Zobrazena je na obrázku 3.8a. Kromě jména je u každého nákupního lístku také datum jeho vytvoření a počet položek, skládající se ze dvou čísel. První značí ty položky, které již byly zaškrtnuté, tedy ty, které máme v košíku. Druhé pak celkový počet položek na daném nákupním lístku. Z kontextového menu, které je možné vyvolat dlouhým stiskem položky, pak bude možné změnit název lístku nebo lístek smazat. V horní části ještě můžeme najít vstupní pole pro jméno nového lístku a tlačítko na jeho vytvoření.

Kliknutím na lístek se dostaneme do druhé aktivity naší aplikace, což bude seznam položek daného lístku. V horní části aktivity jsou umístěny dvě tlačítka, jedno pro návrat zpět a druhé pro přidání nové položky. Pod nimi již bude seznam všech položek daného nákupního lístku. U každé z nich bude ještě zobrazeno množství, které je třeba zakoupit, včetně jednotky, ve které chceme zboží nakupovat. Kliknutím na položku si budeme moci poznačit, že jsme zboží již zakoupili. Jak je vidět na obrázku 3.8b, označení je provedeno přeškrtnutím názvu a přidáním obrázku zelené odrážky na pravou stranu. Z kontextového menu pak můžeme položku editovat nebo smazat.



(a) Seznam nákupních lístků (b) Seznam položek na vybraném lístku (c) Vytvoření/úprava položky

Obrázek 3.8: Návrh rozmístění funkčních prvků aplikace

Vytváření a editace položky bude realizováno v jedné aktivitě. K rozlišení, o kterou akci se jedná, použijeme informaci, která bude přidána k itentu. Rozdíl tedy bude v tom, že při přechodu do příslušné aktivity budeme mít při editaci na začátku k dispozici číslo položky,

která se bude upravovat. V aktivitě, která je zobrazena na obrázku 3.8c, pak budou načteny hodnoty dané položky do příslušných polí, kde je bude možné následně změnit a zase uložit. Kromě názvu se zde nachází ještě volba množství, u které jsou dvě pomocná tlačítka pro rychlé zvětšení a zmenšení hodnoty o jedničku. Dále je pak možné zvolit jednotku pomocí výběrového seznamu nebo využít největšího pole a přidat k položce poznámku.

Jelikož může dojít k uzamčení displeje nebo příchodu telefonního hovoru a přerušení aplikace bez jakékoliv záruky následného obnovení, bude nutné navrhnout ochranu před ztrátou dat. Ta bude fungovat tak, že po zadání názvu položky se při odchodu z aktivity budou data vždy automaticky ukládat do databáze. K tomu použijeme umístění příslušného kódu do metody *onPause*, která je vyvolána vždy při přerušení aktivity. Výjimku bude tvořit pouze stisknutí tlačítka *Zruš*, které povede k navrácení do předchozí aktivity bez toho, aby byla položka uložena.

3.17 Shrnutí

V této kapitole byl popsán návrh mobilní aplikace v četně kompletního teoretického pozadí, které budeme potřebovat při implementaci. Ta je popsána v kapitole 6. Nyní se zaměříme na návrh jedné z rozšiřujících funkcí, a to na propojení mobilní aplikace s webovou stránkou, ze které bude možné na zařízení přidávat nákupní lístky. Více bude problém popsán v následující kapitole.

Kapitola 4

Webová aplikace

Velká většina úspěšných mobilních aplikací v současné době spolupracuje se svou vlastní webovou stránkou, proto se ji pokusíme vytvořit i pro náš projekt. Takováto stránka rozšiřuje funkce dané aplikace nebo alespoň umožňuje spravovat data na internetu. Z toho důvodu ji budeme nazývat webovou aplikací, abychom zdůraznili, že nepůjde pouze o statický web, ale o dynamického správce dat pro úzkou skupinu uživatelů. Je třeba také upřesnit, že se nebude jednat o stejnou webovou aplikaci optimalizovanou pro mobilní zařízení, která byla popisována v sekci 3.1, ale její použití se bude předpokládat na běžných počítačích. Abychom si ulehčili při vývoji práci, bude vhodné použít webový framework.

4.1 K čemu slouží webový framework

Jelikož jsou některé části zdrojového kódu totožné téměř pro každou webovou aplikaci, bylo by zbytečné a neefektivní, kdyby se tyto části musely psát znovu a znovu. Proto vznikly webové frameworky, které zastřešují základní funkčnost a programátorům šetří práci, zpřehledňují kód a zrychlují vývoj. Některé mohou být velmi rozsáhlé a podporovat vývoj i těch největších webů, jiné zase naopak mohou obsahovat jen několik základních knihoven. Pro potřeby našeho projektu byl zvolen oblíbený český framework **Nette**, který bude v následující sekci podrobněji popsán.

4.2 Webový framework Nette

Nette byl vybrán, protože se jedná o jednoduchý opensource framework, který je zároveň výkonný a lze s ním pohodlně vytvářet webové aplikace v PHP. Nette má kolem sebe velkou aktivní komunitu a tedy i zdroje a dokumentaci [5], ze které bude v následujících podkapitolách čerpáno. Mezi hlavní přednosti patří technologie, které eliminují bezpečnostní rizika jako *XSS, CSRF* nebo *session hijacking* a podpora *AJAX / AJAJ, SEO, DRY, KISS, MVC, Web 2.0* a dalších. Dokáže také efektivně pracovat s databázemi jako je například *MySQL*, kterou budeme využívat i v naší aplikaci. Velmi silnou stránkou tohoto frameworku jsou ladící nástroje. Jelikož jazyk PHP dává programátorovi značnou volnost, je také zdrojem těžko odhalitelných chyb. Nette ovšem obsahuje jeden z nejlepších diagnostických nástrojů, které jsou dostupné. Umožňuje nám logovat a rychle odhalit chyby, vypisovat proměnné nebo například měřit čas. Jeho použití je navíc velmi snadné. Mimo to obsahuje i velké množství doplňků, které lze použít ve vlastních aplikacích.

4.3 Architektura Nette

Nette je tvořen nám již známou softwarovou architekturou **MVC**. Ta vzájemně odděluje kód do tří částí. První z nich je **model**, který je datovým, ale zejména funkčním, základem celé aplikace a obsahuje aplikační logiku. Vnitřně si spravuje svůj vlastní stav a navenek nabízí pevně dané rozhraní, přes které je možné k tomuto stavu přistupovat a měnit ho. Model představuje celou vrstvu, která neví o dalších částech architektury. Není tedy pouze samostatnou třídou. Druhou částí je **miew**, neboli pohled. Tato vrstva aplikace má na starosti zobrazení výsledku požadavku. Data získává od modelu a k jejich vykreslování používá šablonovací systém. Poslední částí je **controller**, neboli řadič, který zpracovává požadavky od uživatele a na jejich základě pak volá patřičnou aplikační logiku, tj. model. Zpracovaná data pak předává do pohledu k zobrazení.

4.4 Struktura Nette projektu

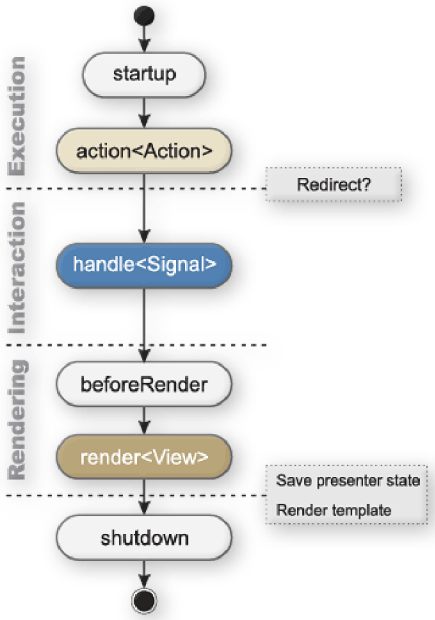
Při vývoji aplikace ve frameworku Nette budeme téměř výhradně pracovat se složkou `app`, která je umístěna v kořenovém adresáři. Dále jsou zde složky s knihovnami, logovacími a dočasnými soubory. Ve složce `app` je umístěna struktura projektu, která se skládá z několika dalších složek, jenž obsahují soubory se specifickou funkčností pro práci frameworku:

- `config/`: Obsahuje soubor `config.neon`, v němž se provádí konfigurace připojení k databázi, registrace služeb nebo třeba nastavení PHP.
- `models/`: V této složce jsou umístěny datové *modely* aplikace. Jsou to třídy, které zapouzdřují a zajišťují komunikaci s datovými úložišti, typicky databázemi. Nejde tedy o architektonický model, který by zajišťoval funkčnost aplikace.
- `presenters/`: Zde se nachází třídy realizující tzv. *presentery*. Ty slouží k propojení jednotlivých částí aplikace – *šablon, komponent* a *modelů*. Obecně je v nich umístěna aplikační logika a ostatní vrstvy volají akce jednotlivých presenterů. Pokud chceme například vytvořit funkci zobrazení detailu produktu, můžeme vytvořit presenter *Product* a v něm akci *show*. Samotnou akci pak provedeme kliknutím na odkaz, který bude mít jako cíl deklarováno `Product:show`. Presenter ovšem nemusí předkládat pouze HTML dokument, ale může to být i obrázek nebo jiný soubor. Konkrétně v našem případě se ale stane to, že `ProductPresenter` požádá model o data, která pak předá do šablony k vykreslení.
- `templates/`: Tento adresář obsahuje další strukturu složek se *šablonami* pro každý presenter kromě *BasePreserteru*, který slouží jako základ, ze kterého ostatní dědí. V šablonách je možné specifikovat grafickou podobu jednotlivých stránek. V Nette také existuje dědění šablon. Je zde tedy jedna nadřazená šablona, `@layout.latte`, ve které mohou být definovány bloky, které pak budou v potomcích přepsány. Jedním z dalších vylepšení ve vypisování šablon jsou makra, která slouží k implementaci složitějších jazykových konstrukcí a umožňují tak provádět zobrazování dat efektivněji.

4.5 Životní cyklus presenteru

Pokud je vyvolána akce presenteru, například kliknutím na odkaz na stránce, volá se metoda `render<Akce>` (například tedy `renderShow`), která se stará zejména o předání dat

do šablony. Jak je ale vidět na obrázku 4.1, je kromě této metody voláno ještě množství dalších. První z nich je metoda `action<Action>`. Ta slouží pro situace, kdy je potřeba provést určitý úkon, jako například přihlásit uživatele nebo zapsat něco do databáze. Dávat takové metodě název *render* by bylo nezvyklé, proto to používá alternativa s názvem *action*. Důležité je ale vědět, že se tato metoda provede jako první. Poté následuje metoda `handle<Signal>`, která, jak název napovídá, zpracovává signály neboli subrequesty. Toho využijeme zejména při práci s komponentami nebo zpracování AJAX požadavků. Před zavoláním metody *render*, která již byla popsána výše, je provedena ještě metoda `beforeRender`. V té můžeme provádět například nastavení šablon nebo předání proměnných společných pro více pohledů. Začátek a konec životního cyklu presenteru je pak dán metodami `startup` a `shutdown`.



Obrázek 4.1: Životní cyklus presenteru ve webovém frameworku Nette [5]

4.6 Návrh propojení s mobilní aplikací

Jak již bylo avizováno v úvodu, cílem vytváření webové aplikace je její propojení s aplikací mobilní za účelem vzájemné spolupráce. Bude tedy třeba vytvořit jistou službu fungující mezi těmito dvěma platformami, která bude realizovat jejich propojení. Možným řešením tohoto problému by mohla být *Oracle Database Mobile Server* [13]. Jedná se o komplexní službu, která zajišťuje synchronizaci a zálohování dat mezi mobilními databázemi na různých platformách a hlavní Oracle databází. Mobilní aplikaci tedy stačí propojit s touto hlavní databází přes příslušné API a o nic víc se není třeba starat. Data se budou automaticky synchronizovat. Problémem tohoto řešení je, že je pod *OTN lincencí*, která umožňuje software využít pouze pro nekomerční účely. Navíc se jedná o zbytečně velmi komplexní řešení v porovnání s rozsahem funkcí, které chceme realizovat pro náš projekt.

Použijeme tedy raději jednodušší variantu v podobě populární open-source databáze

MySQL, ke které budeme přistupovat pomocí PHP skriptů. Vše umístíme na serveru veřejně dostupném z internetu. Z pohledu mobilní aplikace budeme k daným skriptům přistupovat pomocí HTTP protokolu, který můžeme v systému Android využít díky předem připraveným knihovnám. To nám umožní jednoduchým způsobem přenést SQL dotaz v podobě dat, která budou vstupem PHP skriptu, a zároveň v mobilní aplikaci počkat na přijetí vygenerované odpovědi, která bude obsahovat odpověď z databáze.

Ještě je ale třeba zamyslet se nad formátem přenášených dat. Pokud totiž přenášíme složitější struktury, jako například několik řádků tabulky najednou, budeme chtít, aby nedošlo k záměnám sloupců a podobně. Z tohoto důvodu dopředu určíme datovou strukturu tak, aby bylo vždy jasné, co přenáší za informace. K tomuto účelu použijeme **JSON** standard. Práce s daty v tomto formátu je implementována jak v jazyce PHP, který budeme používat na straně webového serveru, tak v Java knihovnách, které můžeme využít při vývoji aplikace pro Android.

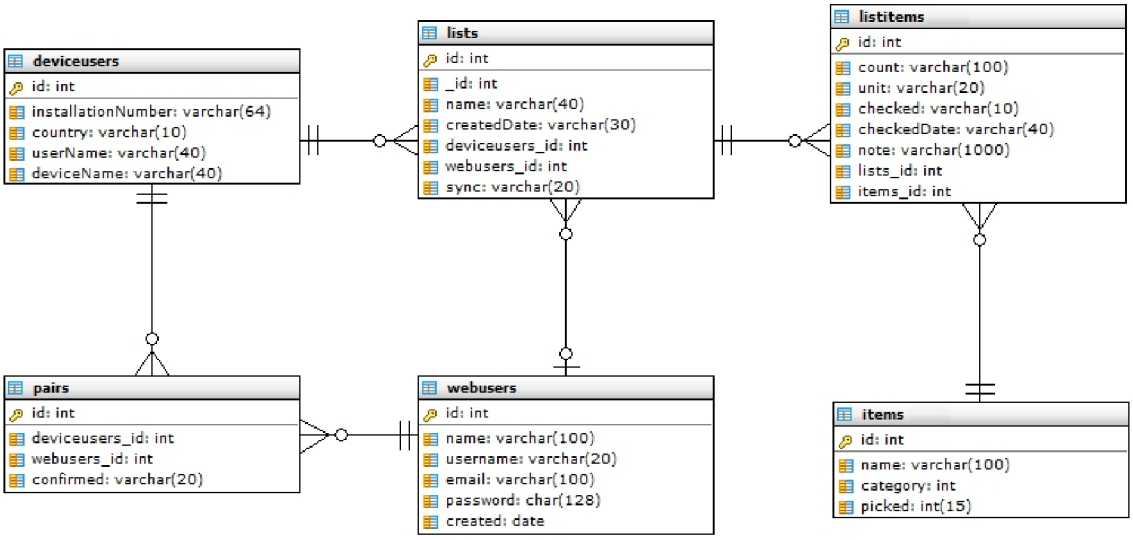
4.7 Uživatelské účty

Jelikož vytváříme dvě propojené aplikace, musíme vytvořit schéma, které bude definovat vztah mezi webovými a mobilními uživateli. Jedním řešením by mohlo být použití společného účtu k oběma aplikacím. Vyplývala by z něho potřeba implementovat přihlašování, ale případně i registraci uživatelů z mobilního zařízení. Tato varianta by se hodila, pokud bychom implementovali kompletní synchronizaci všech dat. V našem případě bylo ovšem zvoleno pro danou situaci více se hodící řešení s oddělenými účty pro obě aplikace. To nám přinese možnost přenášet nákupní lístky z jednoho webového účtu na více mobilních zařízení a zároveň posílat lístky na jedno zařízení z více různých webových účtů. Toto řešení s sebou ovšem přináší komplikace v podobě autorizace uživatelů k zasílání lístků a identifikace zařízení, na které má být lístek zaslán. Budeme tedy hledat jednoduchý způsob, jak by webový uživatel mohl určit zařízení, na které chce poslat lístek. Zároveň budeme ověřovat jeho právo lístek odeslat, a zda s tím uživatel mobilního zařízení souhlasí.

Za tímto účelem byl navrhnut model, kdy budeme mezi sebou uživatele párovat a pouze mezi vzájemně potvrzenými dvojicemi bude možné lístky přenášet. Způsob, kterým by měly být páry vytvářeny, je následující. Nejprve se bude muset uživatel zaregistrovat v obou aplikacích. V té webové běžnou registrací a v mobilní zadáním svého jména a unikátního identifikátoru zařízení. Zde by bylo vhodné použít unikátní číslo generované vnitřní systémovou metodou. Jelikož Android přímo nepodporuje žádnou možnost získání takového čísla, které by bezpečně fungovalo na všech zařízeních, bylo zvoleno řešení, kdy si unikátní identifikátor volí uživatel sám. Jako výhoda se to ukáže hned při dalším kroku, kdy je třeba z webové aplikace najít mobilní zařízení právě pomocí tohoto unikátního jména. A je zřejmé, že se uživatelům budou lépe vyhledávat zařízení se jmény, která si budou moci sami zvolit, než pomocí dlouhých čísel v šestnáctkové soustavě, ve které se nejčastěji takovéto identifikátory nacházejí. Aby bylo párování dokončeno, je třeba ještě návrh webového uživatele v mobilní aplikaci potvrdit. Po úspěšném spárování již bude možné zasílat na mobilní zařízení nové nákupní lístky bez nutnosti dalších potvrzení. Párování by ovšem mělo být možné zrušit z obou stran.

4.8 Návrh datové struktury

Když víme, jakým způsobem mezi sebou budou naše dvě aplikace komunikovat, zbývá tedy ještě vyřešit, jaká data vlastně budeme přenášet a co budeme ukládat v MySQL databázi. Jelikož hlavním účelem webové aplikace je možnost přidávat vzdáleně nové nákupní lístky do mobilního zařízení, nebude třeba evidovat velké množství dat. V návaznosti na systém párování uvedený v předchozí sekci, budou v databázi, jejíž návrh je zobrazen na obrázku 4.2, dvě tabulky uživatelů. První *webusers* pro webové a druhá *deviceusers* pro mobilní. Mezi nimi pak bude existovat společná tabulka *pairs*, která bude tvořit vztah *spárováno*. Samotné potvrzení spárování se pak bude realizovat pomocí atributu *confirmed*.



Obrázek 4.2: Schéma databáze pro MySQL server v *Crow's foot* notaci

Další tabulky již budou tvořit systém ukládání nákupních lístků a položek, který je založen na stejném principu jako v mobilních zařízeních, jenž byl popsán v sekci 3.15. Podstatné je zde navržené propojení lístků (*items*) s oběma tabulkami uživatelů. Rozdíl mezi těmito dvěma relacemi je v tom, že každý lístek může být spojen s jedním uživatelem webové aplikace, ale musí být spojen právě s jedním uživatelem aplikace mobilní. To bylo navrženo proto, aby databáze mohla obsahovat i lísky a položky vytvořené v mobilním zařízení, které jsou z něj odesílány ve chvíli vytvoření, a to pouze pokud existuje připojení k internetu. Nebude tedy garantováno uložení všech lístků a položek na webový server, což ani nebylo záměrem.

Další přínosem, který vyplývá z navrženého databázového modelu, je ukládání počtu výběrů jednotlivých položek mezi všemi uživateli. Ten je opět realizován atributem *picked* v tabulce *items*, ale na rozdíl od mobilní verze databáze na globální úrovni. Tímto způsobem budeme moci získávat cenné informace o oblíbenosti jednotlivých položek mezi všemi uživateli a podle toho v budoucnu případně upravit databázi položek.

4.9 Shrnutí

V této kapitole byl popsán teoretický rozbor a návrh komponenty, která nám umožní zasílat na mobilní zařízení nákupní lístky přes internet. Jedná se pouze o rozšiřující funkci mobilní aplikace, čemuž také odpovídá její rozsah. Nebyl zde uveden ani návrh vzhledu, jelikož se aplikace bude skládat ze tří stránek, které budou realizovat samotnou funkčnost. Na první stránce budou evidována spárovaná zařízení a přidávána nová, na druhé bude možné vytvářet nákupní lístky a odesílat do vybraného zařízení a na třetí pak bude historie. Implementaci této služby můžeme nalézt v kapitole 6. Následující kapitola bude zaměřena na unikátní funkci inteligentního řazení položek na lístku.

Kapitola 5

Inteligentní řazení položek

Hlavní unikátní a inovativní funkcí mobilní aplikace, která by nám měla poskytnout konkurenční výhodu, představuje možnost inteligentně seřadit položky na nákupním lístku tak, aby odpovídaly rozložení zboží v obchodě. Uživatel tak bude moci projít obchodem rychleji a položky na lístku bude mít automaticky v pořadí, jak je bude v obchodě potkávat.

Aby ale bylo možné něco podobného realizovat, musíme někde získat onu mapu rozložení položek v každém obchodě. Možným zdrojem těchto dat by mohly být přímo obchody. Ty zřejmě nebudou chtít tyto informace poskytnout, jelikož rychlé projití zákazníka obchodem není v jejich zájmu. Zaměříme se tedy na jiné možnosti, ze kterých nejlépe vychází metoda, kdy si tato data uživatel vytvoří sám. Aby bylo zadávání dat co možná nejvíce automatické a nevyžadovalo žádnou přídavnou aktivitu oproti běžnému využívání aplikace, budou se data získávat z pořadí zaškrtávání položek na nákupním lístku v průběhu průchodu obchodem. Jediné, co bude muset uživatel učinit navíc, je výběr obchodu, ve kterém nákup provádí.

V ideálním případě tedy budeme mít kolekci několika nákupů ve stejném obchodě, u kterých budeme znát pořadí, v jakém uživatel dával jednotlivé položky do košíku. Z kapacitních důvodů budeme ukládat pouze informace o kategoriích zboží a ne každé jednotlivé položce, kterých by mohlo být na tisíce. Na danou kolekci budeme dále aplikovat algoritmus, jehož vstupem bude nový nákupní lístek s položkami zařazenými do kategorií. Výstupem pak bude nejpravděpodobnější pořadí daných kategorií odvozené z dat nasbíraných během předchozích nákupů. Řešení budeme hledat v oboru zpracovaní přirozeného jazyka, přičemž v následujících podkapitolách budeme čerpat z publikace [14].

5.1 Jazykový model

Hlavním úkolem jazykového modelu při zpracování přirozeného jazyka je určení následujícího slova na základě předchozích. Toho se dá využít při rozpoznávání mluvené řeči, strojovém překladu, ale i v mnoha dalších případech. Pro náš konkrétní případ existuje analogie slov a kategorií zboží, které chceme určovat. Místo modelu jazyka tak budeme mít model obchodu, který vytvoříme z trénovacích dat v podobě sekvencí kategorií místo vět. Díky tomu budeme moci vyčíslit pravděpodobnost sekvence kategorií v závislosti na modelu obchodu a při použití vhodného algoritmu pak najít nejpravděpodobnější posloupnost kategorií pro daný obchod neboli nejpravděpodobnější cestu obchodem.

Po modelu obchodu tedy budeme chtít, aby ocenil pravděpodobnost posloupnosti kategorií $K = \{k_1, k_2 ... k_N\}$, která se může objevit na nákupním lístku. K tomu budeme potře-

bovat znát pravděpodobnost kategorie na každém jednotlivém místě v řetězci. Tu určíme z levého kontextu, neboli z předchozích kategorií v posloupnosti. Matematicky celkovou pravděpodobnost posloupnosti určíme jako

$$P(K) = P(k_1)P(k_2|k_1)P(k_3|k_1k_2)...P(k_N|k_1...k_{N-1}) = \prod_{n=1}^{N} P(k_n|k_1...k_{n-1})$$

Pokud bychom toto chtěli aplikovat na příklad, počítala by se pravděpodobnost založená na předchozích nákupech. Její hodnota by byla složena z pravděpodobnosti, zda prvně bylo nakoupeno zboží například v kategorii pečiva, násobeno pravděpodobností nákupu v kategorii uzenin, za předpokladu předchozího nákupu pečiva, násobeno pravděpodobností nákupu v kategorii mléčných výrobků, za předpokladu, že předtím bylo nakupováno pečivo a následně uzeniny. Takto bychom pokračovali celou sekvencí. Pokud bychom ovšem chtěli takovouto pravděpodobnost v praxi vyjádřit, zjistili bychom, že je to téměř nemožné. Pro množinu $\mathcal{K}$ kategorií o rozměru $|\mathcal{K}|$ existuje pro každou kategorii na n-té pozici $|\mathcal{K}|^{n-1}$ historií. To znamená, že bychom pro vytvoření modelu, který je vlastně tvořen pravděpodobnostmi všech sekvencí, museli spočítat $|\mathcal{K}|^n$ pravděpodobností. I pro malá čísla je to nerealizovatelné množství výpočtů, pokud uvážíme, že model budeme muset občas přepočítávat. A druhým problémem je, že některé kombinace posloupností $k_1...k_n$ se v naší historii průchodů nemusí vůbec vyskytovat. Tento problém řeší [6] rozdělením všech možných historií do menších souborů tak, že všechny historie končící na stejné dvě kategorie budou brány jako ekvivalentní, tj.

$$P(k_n|k_1...k_{n-1}) = P(k_n|k_{n-2}k_{n-1})$$

a potom tedy

$$P(K) = \prod_{n=1}^{N} P(k_n|k_{n-2}k_{n-1}) = P(k_1)P(k_2|k_1)\prod_{n=3}^{N} P(k_n|k_{n-2}k_{n-1})$$

Modelu, který je tvořen takovýmito zkrácenými historiemi, sestávajícími se ze třech po sobě následujících kategorií, říkáme **trigramový model**. Obdobně pak můžeme pro n-tice následujících kategorií vytvořit **n-gramové modely**.

5.2 Oceňování n-gramového modelu

Oceňování pravděpodobností n-gramových modelů je založeno na aproximaci těchto pravděpodobností relativními četnostmi výskytu příslušných n-tic kategorií, objevujících se v trénovacích datech, tedy v minulých průchodech daného obchodu. Pokud budeme mít v systému 25 kategorií, počet všech možných trigramů bude 13 800. Jak již bylo řečeno, většina z nich se v trénovacích datech vůbec neobjeví. Jednoduše proto, že uživatel nikdy takovouto kombinaci kategorií neprošel a dost možná ani v reálném obchodě neexistuje. Při vyčíslování relativní četnosti výskytů bychom pro tyto trigramy dostávali nulovou pravděpodobnost, což by negativně ovlivnilo funkci vyhodnocování nejpravděpodobnějšího pořadí. Tento problém se nazývá **problémem nedostatečných dat**. Jeho řešením může být tzv. **Katzův back-off model**, popsaný v [7]. Ten říká, že pravděpodobnosti chybějících bigramů, resp. trigramů, mohou být odvozeny z relativních četností individuálních kategorií (unigramů), resp. z dříve odhadnutých bigramů.

Předpokládejme, že trénovací data jsou sestavena ze slovníku, který obsahuje celkem $|\mathcal{K}|$ různých kategorií a nechť N_2, resp. N_3, je počet všech bigramů, resp. trigramů. Dále nechť

$N(k_1)$, resp. $N(k_1k_2)$ udává počet, kolikrát se v trénovacích datech vyskytla kategorie k_1, resp. dvojice k_1k_2. A konečně nechť r_2, resp. r_3, udává počet různých dvojic, resp. trojic, kategorií, které se v trénovacích datech vyskytly právě jednou. Potom můžeme vyjádřit odhad pravděpodobností bigramů, které se v tréniknových datech nevyskytly ani jednou, jako r_2/N_2. Nyní můžeme na základě tohoto předpokladu stanovit vztah pro ocenění pravděpodobností všech bigramů jako

$$P(k_2|k_1) = \begin{cases} \left(1 - \frac{r_2}{N_2}\right) \frac{N(k_1k_2)}{N(k_1)}, & \text{pokud } N(k_1k_2) > 0 \\ \left(\frac{r_2}{N_2}\right) \frac{N(k_2)}{K_2(k_1)}, & \text{pokud } N(k_1k_2) = 0, \end{cases}$$

kde $K_2(k_1)$ je normalizační faktor. Zajišťuje, aby suma pravděpodobností přes všechny kategorie k_1 byla rovna 1. Obdobně můžeme stanovit i vztah pro trigramy, kde se nově bude vyskytovat počet všech trojic $k_1k_2k_3$ v trénovacích datech $N(k_1k_2k_3)$ a počet všech odlišných trojic r_3, které se v trénovacích datech vyskytují právě jednou

$$P(k_3|k_1k_2) = \begin{cases} \left(1 - \frac{r_3}{N_3}\right) \frac{N(k_1k_2k_3)}{N(k_1k_2)}, & \text{pokud } N(k_1k_2k_3) > 0 \\ \left(\frac{r_3}{N_3}\right) \frac{P(k_3|k_2)}{K_3(k_1k_2)}, & \text{pokud } N(k_1k_2k_3) = 0. \end{cases}$$

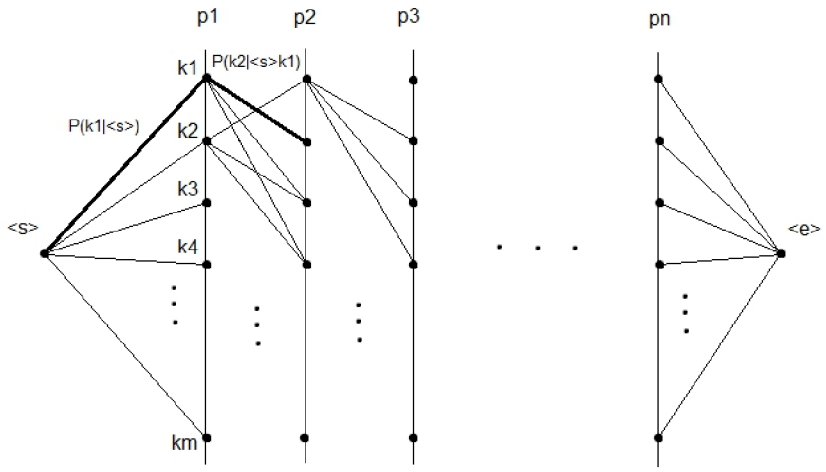
Přitom $K_3(k_1k_2)$ je opět normalizační faktor a neexistuje-li výskyt posloupnosti $k_1k_2k_3$, můžeme k výpočtu bigramu využít předchozího vztahu.

5.3 Vyhodnocení lístku modelem

Vyhodnocení promluvy modelem se v oblasti zpracování přirozeného jazyka rozumí určení pravděpodobnosti, s jakou byla daná promluva O generována vypočítaným modelem λ, tedy $P(O|\lambda)$. Jedním z algoritmů, který se k tomu používá, je **forward-backward**, který provádí rekurzivní výpočet odpředu nebo odzadu generované posloupnosti. My ovšem místo výpočtu pravděpodobnosti jedné posloupnosti potřebujeme spočítat nejpravděpodobnější pořadí kategorií položek, které uživatel zadá. Toho se budeme snažit dosáhnout vypůjčením si principů z metody **token propagation**. Výsledkem našeho snažení bude algoritmus jehož vizualizace je na obrázku 5.1.

Vstupem algoritmu bude kromě modelu obchodu také seznam položek, resp. jejich n kategorií, na nákupním lístku. Ty chceme na výstupu seřadit tak, aby co možná nejvíce odpovídaly danému obchodu, tedy najít takové pořadí daných kategorií, které bude při vyhodnocení modelem dávat největší pravděpodobnost. Celý algoritmus pak bude založen na rozesílání tokenů mezi uzly, které představují kategorie a na obrázku 5.1 jsou zobrazeny ve svislých liniích. V každém okamžiku může existovat pouze m tokenů, tedy pro každou kategorii jeden. Princip algoritmu pak bude založen na propagování každého tokenu do všech následující uzlů, které se provede $n + 1$ krát. Jelikož budeme předpokládat, že uživatel nakoupí zboží jedné kategorie v obchodě na jednom místě, můžeme sjednotit na lístku všechny položky stejných kategorií k sobě a kategorii uvažovat pouze jednou. V algoritmu tak budeme počítat jen s unikátními kategoriemi na lístku. Z toho vyplývá, že počet položek se bude rovnat počtu kategorií $n = m$, neboli budeme hledat pořadí pouze pro unikátní kategorie. Každý přechod mezi dvěma uzly bude ohodnocen pravděpodobností určenou trigramem, případně bigramem, z modelu obchodu. Na obrázku 5.1 můžeme vidět

příklad, který je vyznačen tučně. Ve druhém cyklu se zde bude přecházet z uzlu k_1 do k_2 a pokud víme, že daný token přišel z uzlu <s>, pak použijeme pravděpodobnost trigramu $P(k_2|<s>k_1)$. Tou vynásobíme celkovou kumulativní pravděpodobnost tokenu, kterou v sobě nese už od počátečního uzlu <s>, ve kterém celý algoritmus začíná s jedním tokenem s pravděpodobností jedna.



Obrázek 5.1: Algoritmus pro výpočet posloupnosti s nejvyšší pravděpodobností

Z tohoto principu vyplývá, že se nám po každém cyklu v uzlech nashromáždí několik tokenů, které je třeba zredukovat. To provedeme tak, že zrušíme všechny tokeny kromě jednoho, který bude mít nejvyšší kumulativní pravděpodobnost. V závěrečném kroku algoritmu jsou všechny zbývající tokeny poslány do uzlu <e>, kde opět provedeme redukci. Zůstane nám zde pouze jediný token, který prošel postupně kategoriemi tak, že vyhodnocení modelem obchodu má nejvyšší pravděpodobnost ze všech kombinací. Aby bylo možné danou sekvenci určit, je ještě potřeba v každém tokenu vytvořit zásobník, do kterého se budou ukládat kategorie, kterými prošel. Je třeba ale také poznamenat, že budeme kromě méně pravděpodobných tokenů rušit také ty, které se dostanou do kategorie, kterou již prošly. Kdybychom tak nečinili, mohly by nám vzniknout i posloupnosti, ve kterých by se některé kategorie opakovaly a nebylo by je pak možné použít jako klíč k jednoznačnému řazení položek na nákupním lístku.

5.4 Složitost algoritmu

U algoritmů je vždy dobré vyčíslit časovou složitost, s jakou se budou vykonávat. Abychom mohli porovnat výkonnost algoritmu v závislosti na velikosti vstupních dat, bude dobré určit jeho asymptotickou časovou složitost. Náš algoritmus pro vyhodnocování lístků modelem bude provádět tolik cyklů propagace tokenů, kolik je unikátních kategorií na nákupním lístku, tedy n. V každém takovém cyklu se pak bude v nejhorším případě každý z n tokenů duplikovat právě $n-1$ krát. Z toho vyplývá, že za nejhorších okolností může proběhnout operace výpočtu nového tokenu $n*(n-1)*n$ krát. Asymptotická složitost tedy bude $O(n^3)$, což představuje *kubickou* složitost.

Druhý algoritmus, který používáme pro výpočet modelu obchodu, bude mít asymptotickou složitost podobnou. Musíme zde totiž spočítat všechny trigramy, které v systému mohou existovat. Jejich počet N bude záležet na celkovém počtu kategorií c, do kterých jsou přiřazeny položky z trénovacích dat pro daný obchod. Potom bude celkový počet trigramů v modelu $N = c * c * (c - 1)$, jelikož kategorie se nemohou opakovat, ale na pozici historie musíme počítat ještě se startovním symbolem. Asymptotická složitost algoritmu pro výpočet modelu obchodu tedy bude také $O(c^3)$.

Složitosti, které jsme pro algoritmy určili, nám naznačují, že půjde o časově náročné výpočty. V našem případě se naštěstí bude jednat pouze o malá vstupní data, a tudíž bychom stále měli být schopni výpočet provést. Navíc algoritmus pro výpočet trigramů nebude třeba provádět tak často, pouze v okamžiku, kdy bude do trénovacích dat přidána nová sekvence kategorií. Krom toho máme ještě v záloze optimalizace, které se mohou objevit později během samotné implementace algoritmů a pozitivně nám tak ovlivnit rychlost celého výpočtu.

5.5 Shrnutí

V této kapitole byl probrán návrh a zároveň teoretické pozadí algoritmů pro inteligentní řazení položek. Jejich implementace bude popsána v následující kapitole, konkrétně v sekci 6.5.

Kapitola 6

Implementace a vyhodnocení

V této kapitole bude popsána implementace návrhů, které byly rozebrány v předchozích třech kapitolách. Kromě problémů, se kterými jsme se setkali při vývoji, se zaměříme především na to, jaká je výsledná podoba jednotlivých komponent a jak jsou mezi sebou propojeny. U inteligentního řazení se navíc pokusíme změřit dobu vykonávání jednotlivých algoritmů a porovnat je s předem odhadnutou časovou složitostí. Na závěr zanalyzujeme některá data, která byla v průběhu nasazení aplikace nashromážděna ve webové databázi.

6.1 Získávání zpětné vazby

V sekci 2.3 jsme deklarovali, že během vývoje budeme k testování našich domněnek a funkčních prototypů využívat early adopters. Z jejich zpětné vazby, která byla vždy po testování vyhodnocena, bylo rozhodováno o následujícím směru vývoje, úpravách stávající verze a přidání nových funkcí.

Pro náš projekt bylo zajištěno pouze několik opravdových early adopters, což je malý počet a pro reálný startup by byl pravděpodobně nedostačující. I tak bylo získáno velké množství cenných informací a nápadů, na které by obyčejní uživatelé zajisté nepřišli. Zmíníme se o nich v popisu vývoje jednotlivých částí. V praxi tak bylo ověřeno, že mít tyto lidi k dispozici pro testování je velkou výhodou a žádném startupu by neměli chybět. Rozdíl oproti obyčejným uživatelům nebo testerům je totiž zásadní.

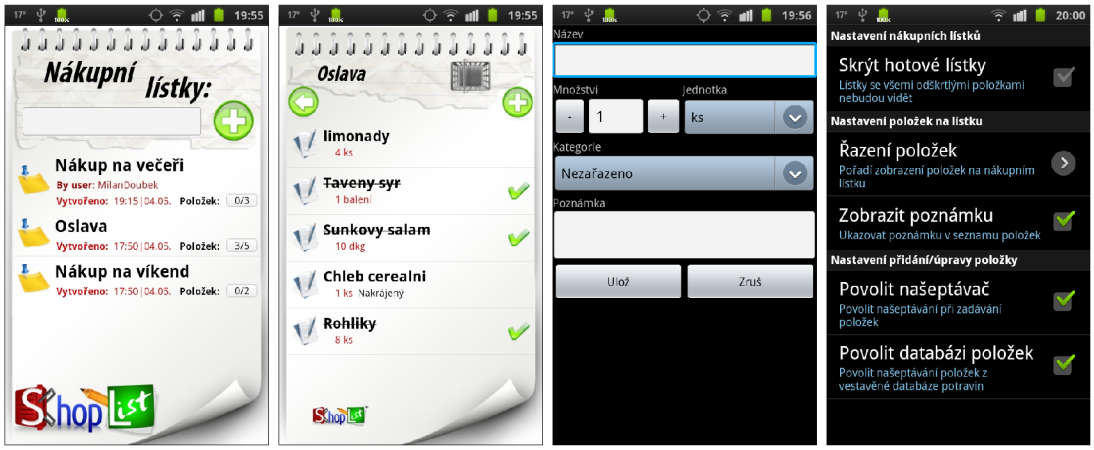
Aby byla zpětná vazba co nejkvalitnější a získali jsme co možná nejvíce informací, byla zde snaha získávat tyto údaje i z jiných zdrojů, než pouze od early adopters. Prvním z nich byli obyčejní uživatelé a testeři. Ti sice neposkytli tak kvalitní zpětnou vazbu, ale dokázali jistou měrou přispět ke zkvalitnění aplikací. Například odhalováním překlepů v textech a jiných drobných chyb. Druhou alternativní možností získávání zpětné vazby bylo prezentování projektu na různých akcích, seminářích a jiných událostech. Zde je možné se setkat s odborníky z praxe, představit jim své záměry a konzultovat je. V našem případě šlo o účast na semináři **Jak založit vlastní firmu**, setkání s inverstory na **MIC Minutes** a účast ve startapovém inkubátoru **StarCube** v rámci školního předmětu **Inovace a podnikání v IT**, včetně prezentace projektu na závěrečné **StarCubeShow**. Na každé z těchto akcí byl projekt představen ve své aktuální podobě a byla k němu získána velmi kvalitní zpětná vazba, která v mnoha případech zásadně ovlivnila další směr vývoje.

6.2 Mobilní aplikace

V této sekci bude popsán výsledek několikaměsíčního vývoje, po kterém byla vytvořena mobilní aplikace. Zmíníme se také o několika důležitých verzích, které vznikly během jednotlivých iterací.

Podoba výsledné verze aplikace je zobrazen na obrázcích 6.1. Dopředu je třeba říci, že grafika pochází z externího zdroje a její vytvoření nebylo součástí této práce. Ta se skládala pouze z jejího zpracování, nastříhání a rozmístění po displeji dle návrhu ze sekce 3.16. Mnohem důležitější a náročnější ovšem bylo udělat její optimalizaci pro různá zařízení, na kterých běží operační systém Android. Bylo přitom postupováno podle zásad popsaných v sekci 3.14. Výsledkem této práce je možnost použít aplikaci na všech současných mobilních telefonech s displeji různých velikostí a rozlišení, aniž by docházelo k rozbití její grafické podoby. Díky tomu je v současné době dostupná pro téměř tisíc různých mobilních zařízení, a to ve dvou jazycích.

Výsledná podoba aplikace vychází z funkčního návrhu ze sekce 3.16 a po implementaci se kromě přidání lepší grafiky výrazně nezměnila. Můžeme zde však zaznamenat několik změn, které vyplynuly z vývoje. Na obrázku 6.1a je například u prvního nákupního lístku vidět řádek označující uživatele, který ho vytvořil. Tento řádek byl přidán z důvodu upozornění na to, že lístek nebyl vytvořen v tomto zařízení, ale byl sem poslán z webové aplikace, a je tedy třeba zobrazit jméno autora. Dále je možné si na obrázku 6.1c všimnout přidání vysouvacího seznamu kategorií, který je možné použít při vytváření nových položek. Je zde i jedna úplně nová aktivita, která je zobrazena na obrázku 6.1d. Ta slouží jako rozhraní pro nastavení a je dostupná z menu v hlavní aktivitě s nákupními lístky. Vychází ze základní šablony *Preferences*, kterou můžeme v systému Android k vytvoření nastavení použít. Uživatel si tak může přizpůsobit prostředí aplikace podle svých potřeb, jako je třeba zapnutí skrývání kompletně vyplněných lístků, výběr řazení položek na lístku nebo vypnutí našeptávače. Žádná z těchto funkcí nebyla součástí návrhu, ale jejich přidání vyplynulo ze zpětné vazby od early adopters a uživatelů. Proto byly postupně implementovány v nově vznikajících verzích aplikace během jednotlivých iterací.



(a) Nákupní lístky (b) Seznam položek (c) Nová položka (d) Nastavení

Obrázek 6.1: Výsledný vzhled mobilní aplikace

Jakmile byla vyladěna první verze aplikace, která poskytovala pouze základní funkčnost,

bylo rozhodnuto o jejím umístění na Google Play market. To zejména kvůli získání prvních skutečných uživatelů a v lepším případně nových early adopters. Před vystavením aplikace bylo třeba vytvořit vývojářský účet a aplikaci patřičně upravit. To nejen z pohledu kódu, ale také optimalizovat její grafické rozhraní. Následně musel být vygenerovaný instalační *apk* soubor, který byl určen k vystavení, digitálně podepsán. K tomu však není nutné mít certifikát podepsaný certifikační autoritou. Stačilo tedy využít nástroj v IDE Eclipse, který pro nás vytvořil nové klíče a provedl samotné podepsání. Následně už byl jen vyplněn popis a nahrány obrázky aplikace na vlastní stránku marketu a vystavení bylo kompletní.

Aplikace zaznamenala na marketu velmi dobrý start, kdy byla během prvního měsíce stažena více než čtyři tisíce krát. Zejména díky její bezchybovosti a příjemnému uživatelskému rozhraní ji uživatelé udělili průměrné hodnocení 4,8 z 5, což je nejlepší hodnocení mezi nákupními lístky. Díky tomu a velké stahovanosti se aplikace dostala i do TOP 15 nových bezplatných aplikací na celém marketu a do TOP 5 nejlepších aplikací v kategorii *Shopping*.

Na první verzi bylo navázáno aktualizací, která přinesla nové funkce a vylepšení, zejména pak takové, které vyplynuly ze zpětné vazby od early adopters a uživatelů. Mezi hlavními změnami byl přidán našeptávač propojený s předpřipravenou databází položek. Její vývoj byl specifický, a proto ji podrobněji popíšeme v následující sekci 6.3. Dále pak byly přidány kategorie a vytvořena nová aktivita implementující možnost úpravy základního nastavení aplikace. Do té byla přidána možnost změny řazení položek na nákupním lístku, z nichž nejvíce žádanou metodou bylo posunování již zaškrtnutých položek na dno lísku v reálném čase. Ač byla aktualizace přijata kladně, objevilo se v ní několik chyb, které nebyly objeveny během testování. To mírně zhoršilo celkové průměrné hodnocení aplikace, což lehce snížilo zájem uživatelů a počet stažení již nerostl takovým tempem. Přesto se počet instalací do současné doby dostal přes hodnotu sedmi tisíc. Aktivních uživatelů, kteří mají aplikaci stále nainstalovanou ve svém zařízení, je okolo čtyř tisíc.

Nejnovější verze aplikace, která bude obsahovat možnost výběru obchodu, inteligentní řazení a propojení s webovou aplikací, zatím na market umístěna nebyla. Je to kvůli tomu, že stále probíhá její testování a ladění, ale zejména z důvodu hledání optimálního modelu, jak aplikaci monetizovat. Ostatní vývojáři své aplikace s podobnými funkcemi nabízí běžně jako placené nebo minimálně omezené reklamou. V budoucnu tedy budeme po aplikaci také požadovat, aby generovala nějaký zisk.

6.3 Vestavěná databáze položek

Jednou z funkcí, která by měla zrychlit zadávání položek do mobilního zařízení, je vestavěná databáze zboží propojená s automatickým našeptávačem. Jeho funkce je taková, že pokud v aktivitě přidání nové položky zadáme první dvě písmena z jejího názvu, aplikace nám automaticky nabídne rozbalovací seznam se všemi řádky z vestavěné databáze, které obsahují slovo začínající na tyto dvě písmena. Z těch je pak možné vybrat požadované zboží, které chceme přidat na lístek, a ušetřit tak čas jinak strávený vypisováním názvu na klávesnici.

Abychom mohli tuto funkci realizovat, bylo potřeba zajistit databázi zboží, převážně pak potravin, které by sloužily uživateli jako nápověda. Řešení bylo nalezeno v použití databáze ze zaniklého projektu internetového obchodu s potravinami, ve které se nacházelo celkem přes pět tisíc položek zařazených do dvaceti pěti kategorií.

Dříve, než ji bylo možné použít, bylo potřeba upravit názvy jednotlivých položek. Ty totiž často obsahovaly i název výrobce a kvantitu, ve které byly prodávány. Jejich úprava

nebyla složitá a skládala se z napsání několika *shell* skriptů, které názvy položek ořezaly o zbytečné informace. Po této úpravě ovšem v databázi vznikly redundantní záznamy, které bylo potřeba eliminovat. Využití zde našel algoritmus pro výpočet **Levenshteinovi vzdálenosti**, pomocí kterého byl vypočítán rozdíl mezi každými dvěma řetězci s názvy položky. V případě, že se od sebe lišily pouze jedním nebo vůbec žádným písmenem, byla jedna z nich odstraněna. Po této a finální ruční korekci zbylo v databázi přes tři a půl tisíce položek.

Posledním problémem k vyřešení bylo dostat takto obsáhlou databázi do aplikace a propojit ji s našeptávačem, který jako vstup bere pole řetězců. Převést data do souboru ve formátu *XML* nebo jemu podobnému zde nebylo možné, jelikož by docházelo k dlouhé časové prodlevě zapříčiněné načítáním dat do programu při každém otevření aktivity pro přidávání nové položky. Druhou možností bylo uvést všechny položky přímo v programu, což je ale programátorsky nekorektní krok. Navíc by to mělo za následek vznik problémů při přidávání nových položek zadaných uživatelem. Nejlepší volbou se tedy ukázalo použití databáze *SQLite*, ze které mohou být načítány operativně pouze ty položky, které mají být v aktuální chvíli zobrazeny.

Aby byl našeptávač připraven k okamžitému použití, je potřeba mít hned po nainstalování aplikace tuto databázi k dispozici. Problém ovšem je, jak ji naplnit položkami. Díky vlastnostem popsaným v sekci 3.10 není problém při prvním spuštění aplikace provést jistý kód. Mohli bychom tedy spustit velký *SQL* skript, který by proběhl pouze jednou a naplnil databázi položkami. Obecně se ovšem spouštění velkého množství databázových dotazů najednou nedoporučuje, protože může docházet k chybám, nehledě na velké výpočetní zatížení systému. Mnohem efektivnějším způsobem je vložit předvytvořenou a předem naplněnou databázi do instalačního souboru aplikace jako binární soubor a při prvním spuštění ji zkopírovat na patřičné místo do systému. Hned je tak připravena k použití a systém je jen minimálně zatížen.

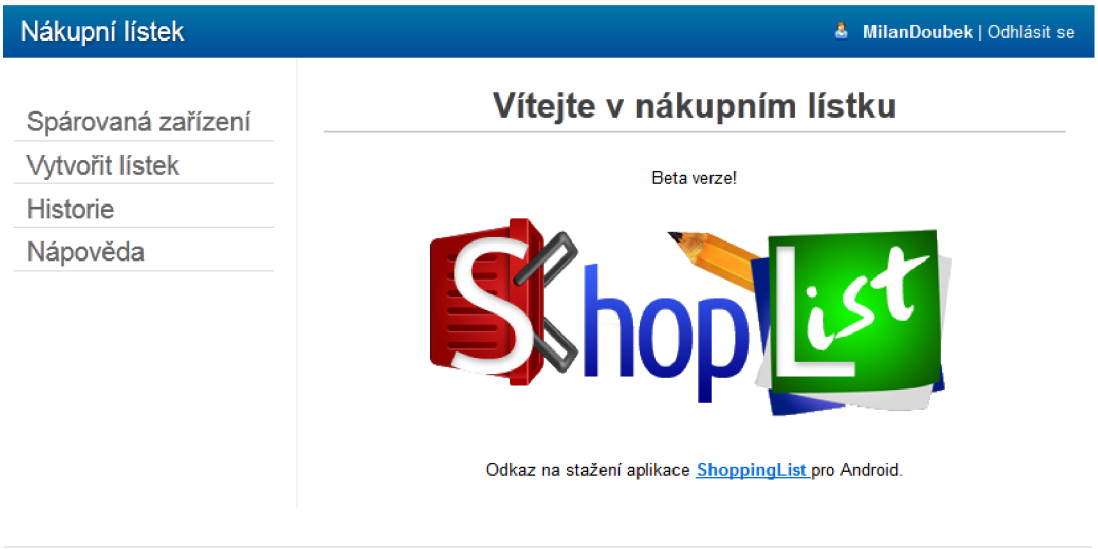
V aplikaci tedy máme předvytvořenou databázi tří a půl tisíce položek, které jsou nabízeny pomocí našeptávače. Pokud by si uživatel z této databáze nevybral, je samozřejmě možné zadat i vlastní položku, která je následně přidána mezi ostatní a při dalším vytváření je již nabízena spolu s ostatními. V aplikaci zároveň funguje omezení, které zabraňuje vzniku duplicit. Pokud by se uživateli povedlo zadat název, který již v databázi existuje, nebude vytvořena položka nová, ale jako zdroj dat se využije ta již existující.

Nyní, když máme vytvořen takovýto model s trvale uloženými položkami, není pro nás problém si u každé z nich pamatovat kategorii, kterou můžeme při vybrání dané položky v našeptávači předvyplnit do příslušné kolonky, a ušetřit tak uživateli práci. Dále je také u každé položky ukládán a aktualizován počet jejích použití na nákupních lístcích. Tento údaj využíváme k určení pořadí zobrazení v našeptávači. Jakmile tedy jednou položku vybereme, bude se nám příště zobrazovat na předních příčkách v závislosti na stavu ostatních položek.

6.4 Webová aplikace a její zakomponování

Webová aplikace vznikla jako jedna z přídavných funkcí k aplikaci mobilní a podle toho k ní bylo také přistupováno. Vychází ze základní šablony frameworku Nette a prozatím pro ni nebyla obstarána odpovídající grafika. Ta bude získána od stejného externího zdroje, ze kterého pochází grafika pro mobilní aplikaci, aby byly obě v souladu. Webová aplikace zatím nebyla podrobena ani žádnému testování, tudíž se vývoj nachází zatím stále v první iteraci. Výslednou podobu webové stránky můžeme vidět na obrázku 6.2, která zobrazuje úvodní stránku hned po přihlášení.

Jak bylo naplánováno v návrhu, celá aplikace je zabezpečena a k jejím funkcím je možné se dostat až po přihlášení. To bylo implementováno pomocí vestavěných prostředků frameworku Nette. Ten například standardně podporuje uložení hesel v databázi v zahashované podobně nebo snadné zabezpečení přístupu nepřihlášeného uživatele ke skrytým stránkám pomocí zadání jejich adresy, včetně automatického přesměrování na přihlašování. Registrace není v současné době nějak omezena a prakticky kdokoliv se může registrovat na základě jména, přezdívky, emailu a hesla. Jelikož k tomu zatím nebyl důvod, email není ani ověřován zasíláním kontrolní zprávy se zpětným odkazem pro potvrzení registrace. Protože je do budoucna v plánu aplikace zmonetizovat, je pravděpodobné, že se tento model změní a registrace budu umožněna pouze vybraným uživatelům.

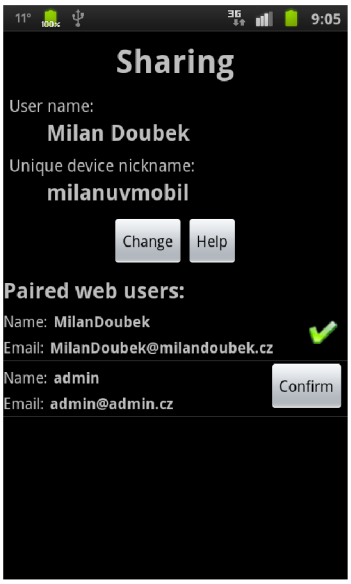


Obrázek 6.2: Úvodní stránka webové aplikace po přihlášení

Jak je vidět na obrázku 6.2, první položka menu odkazuje na stránku, ze které je možné odeslat žádost o spárování s mobilním zařízením. Jak bylo uvedeno v návrhu v sekci 4.7, zařízení jsou identifikována podle unikátního názvu, které musí uživatel webové aplikace znát, aby si mohl svůj účet se zařízením spárovat. Kromě toho se na této stránce také nachází tabulka s již vytvořenými páry. Podstatným sloupcem tabulky je atribut určující, zda již bylo spárování potvrzeno ze strany mobilního uživatele. Dokud tak nebude učiněno, nepůjdou na toto zařízení posílat lístky. Párování je možné ze stránky, která je dostupná pod druhou položkou menu. Uživatel zde po vyplnění názvu lístku a výběru zařízení, se kterým bylo vytvořené potvrzené spárování, může vytvářet seznam položek a následně jej do tohoto zařízení odeslat. Poslední položkou menu je historie nákupních lístků. Zde po výběru zařízení můžeme najít seznam nákupních lístků, které na něj byly odeslány. Kromě nich jsou zde i lísky vytvořené v zařízení. Není ale garantováno, že jsou to všechny lístky, které kdy byly vytvořeny. Pokud totiž není dostupný internet, lístek se ze zařízení neodešle. U každého z nich je pak ještě zobrazen status, udávající zda byl lístek stažen do zařízení nebo teprve na stažení čeká. V přídadě, že se jedná o lístek z mobilního zařízení, je tato informace zobrazena i ve statusu.

Pro nastavení sdílení na straně mobilních zařízení byla vytvořena speciální aktivita,

která je zobrazena na obrázku 6.3. Můžeme zde vidět dva řádky pro libovolné uživatelské jméno a unikátní jméno zařízení, které ho bude jednoznačně identifikovat. Pomocí připojeného tlačítka lze tyto hodnoty kdykoliv změnit. Aby ale bylo možné měnit i jméno zařízení, potřebujeme jiný jedinečný identifikátor, který nám bude zařízení reprezentovat ve webové databázi. K tomu byl použit atribut *intallationNumber* zobrazený na obrázku návrhu datové struktury webové databáze 4.2, se kterým bylo původně počítáno pouze jako s přídavnou informací o zařízení. Toto číslo, které se spočítá pouze jednou při první spuštění, je jedinečné pro každou instalaci. Aplikace pak již při veškeré komunikaci vystupuje pod tímto číslem a název zařízení zadaný uživatelem slouží pouze pro možnost snazšího vyhledání zařízení. Aby nedošlo ke ztrátě tohoto čísla při aktualizaci, uložíme ho jako perzistentní data typu *shared preferences*, což nám zaručí, že dokud uživatel neodinstaluje aplikace nebo nesmaže ručně její data, pak bude jeho účet zachován. V opačném případě mu bude vygenerováno nové instalační číslo a on si k němu zvolí nový unikátní název zařízení. O své nákupní lístky však nepřijde. Do budoucna je v plánu ve webové aplikaci vytvořit funkci umožňující zasílání lístků z historie. Webový uživatel tedy bude moci po spárování se svou novou instalací odeslat do zařízení lístky z instalace minulé.



Obrázek 6.3: Aktivita pro nastavení sdílení v mobilní aplikaci

Druhou částí aktivity, která je na obrázku 6.3, je seznam spárovaných webových uživatelů. Ti se mohou nacházet ve dvou stavech. Pokud bylo požádáno o spárování, které ještě nebylo potvrzeno, je v příslušném řádku zobrazeno tlačítko *Confirm*. Po jeho stisknutí dojde k potvrzení spárování a na řádku se objeví zelená odrážka. Od tohoto okamžiku již mohou být na zařízení zasílány nové lísky a aplikace automaticky každých 30 vteřin kontroluje, zda pro dané zařízení nejsou nějaké nové na serveru ke stažení. Pro možnost okamžité kontroly stavu bylo do menu v hlavní aktivitě přidáno speciální tlačítko, pro okamžité stažení nákupních lístků. Ty jsou po stažení uloženy do databáze telefonu s jediným rozdílem v tom, že je vyplněn atribut *user* v tabulce *lists*, o čemž jsme se již zmiňovali v předchozí sekci. Vhodné by také bylo, aby se při objevení nového lístku na serveru v zařízení vytvořila notifikace v horní liště, která by uživatele upozornila i v případě, kdy aplikace není zapnutá. Na to by ovšem bylo třeba vytvořit službu, viz sekce 3.8, která by po celou dobu běžela v pozadí a

kontrolovala stav serveru. Ve chvíli, kdy by se tam objevil nový lístek ke stažení, vyvolala by vytvoření notifikace. Vzhledem k povaze aplikace byla tato funkce vyhodnocena jako zbytečné zatížení zařízení a prozatím nebyla implementována.

V celku je tedy webová aplikace již plně funkční. Lísky jdou posílat na libovolné zařízení připojené k internetu, které je zároveň snadné nalézt. Zbývá tedy pouze úprava vzhledu a vyladění uživatelského rozhraní, na čemž se budou podílet early adopters, kterým bude aplikace předložena k otestování. Z pohledu mobilní aplikace bylo propojení provedeno nejjednodušším možným způsobem s přihlédnutím na bezpečnost. Veškerá komunikace s webovou aplikací byla realizována pomocí vláken, jejichž princip byl vysvětlen v sekci 3.12. Pro připojení na server je vždy vytvářeno nové vlákno, které zaštiťuje veškerou komunikaci. Aplikace přitom běží dál nezávisle na tomto novém vlákně a po skončení je její stav náležitě upraven podle dat přijatých z webového serveru. Díky tomuto přístupu je ošetřena možnost zamrznutí při nedostupnosti serveru nebo stahování většího objemu dat.

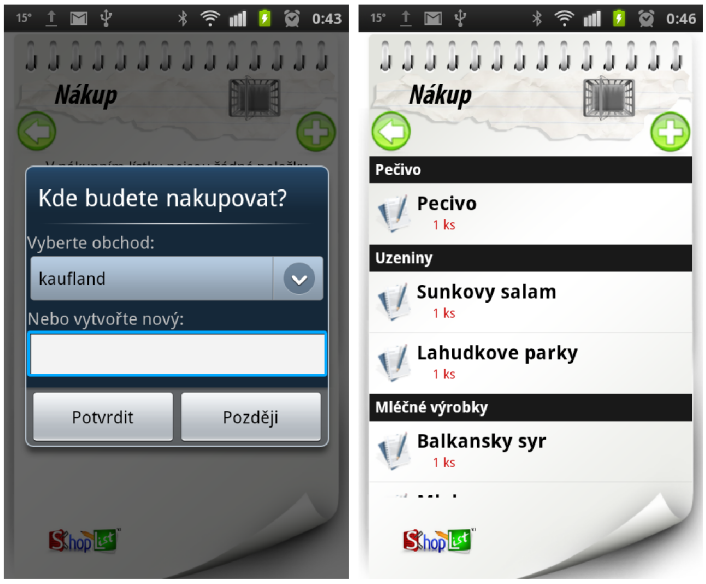
6.5 Inteligentní řazení položek

Implementace funkce inteligentního řazení probíhala dle návrhu v kapitole 5. Prvotní myšlenka pracovat s trigramovým modelem byla ovšem přehodnocena a bylo využito pouze modelu bigramového. To z obavy z vyšší náročnosti výpočtu, ale zejména kvůli principu, který z této volby vyplývá. Pokud totiž budeme pracovat s bigramovým modelem, vybíráme následující kategorii pouze podle jedné předchozí. Řazení tedy bude probíhat stylem, kdy například po kategorii pečiva bude zařazena ta kategorie, kterou uživatel v minulosti nejčastěji navštívil, potom co nakoupil pečivo. Závěr navíc přinesl uspokojivé výsledky, proto bylo od implementace trigramového modelu zcela upuštěno.

Novou metodu řazení bylo třeba také zakomponovat do aplikace. Jelikož už byly v předchozích verzích realizovány jiné řadící metody, stačilo pouze přidat jednu novou. Bylo ovšem třeba vyřešit, jak bude uživatel vybírat a zadávat obchod, ve kterém chce nakupovat. Jak je vidět na obrázku 6.4a, problém byl vyřešen dialogovým vyskakovacím oknem s jedním výsuvným seznamem a jedním vstupním textovým polem. Uživatel tedy může vybrat dříve uložený obchod, nebo zadat nový. Tato výzva bude při zapnutém inteligentním řazení zobrazena vždy při přechodu na nový nákupní lístek, a to opakovaně, dokud uživatel obchod nevybere. Odkaz na vyvolání této výzvy je také umístěn v menu, aby měl uživatel možnost obchod změnit v případě potřeby. Vlastním výběrem je pak zvoleno, který model se bude používat na řazení položek na lístku, ale zároveň také to, ke kterému obchodu bude přiřazena sekvence vzniklá po provedení nákupu. Na obrázku 6.4b je pak zobrazeno zvýraznění jednotlivých kategorií pomocí oddělovačů, které bylo přidáno pro větší přehlednost.

Poté, co je zvolen obchod, může být vypočítán model dle algoritmu navrženého v 5.2. Ten je počítán z historie nákupů v daném obchodě, a proto je pro lístek spuštěn vždy pouze jednou. Druhý algoritmus, který byl popsán v sekci 5.3, nám z modelu vypočítá klíč, podle kterého následně budou položky na lístku seřazeny. To ale znamená, že tento výpočet musí být spouštěn po přidání každé položky. Řešením by mohlo být i vyvolání seřazení položek až po stisku tlačítka. To by ovšem nebylo tak přirozené, jako automatické zařazování položek na správná místa hned po jejich vytvoření. Vzhledem k tomu, že tento algoritmus pracuje pouze s kategoriemi a ne se samotnými položkami, byla implementována optimalizace, kdy nový klíč pro řazení počítáme pouze, pokud je přidána položka s kategorií, která se na lístku ještě nevyskytuje. V opačném případě můžeme využít klíč vypočítaný dříve, který obsahuje všechny aktuální kategorie a můžeme podle něj tedy položky seřadit.

Výsledná úspěšnost této řadící metody je velmi dobrá. Pokud uživatel alespoň jednou



(a) Výběr obchodu (b) Zvýraznění kategorií

Obrázek 6.4: Implementace inteligentního vyhledávání

projde obchodem, zaškrtá všechny položky a následně vytvoří lístek se stejnými kategoriemi, metoda je seřadí přesně do té posloupnosti, ve které byly zaškrtnuty při prvním průchodu. S počtem nákupů se pak zvětšuje přesnost metody v podobě různých kombinací, které je schopna správně seřadit. Problém, který se ale během používání této metody může objevit, spočívá v tom, že při vytvoření dlouhé sekvence kategorií, která se objevila v historii pouze jednou, není algoritmus schopen dohledat spojitost mezi krajními kategoriemi. To vyplývá z principu, kdy se při řazení spoléháme pouze na jednu, případně dvě předchozí kategorie. Tento problém řeší větší množství trénovacích dat v podobě provedených nákupů. Potom již algoritmus zvládne najít cestu i mezi krajními kategoriemi. Dále se může stát, že se na lístku objeví nová kategorie, která se zatím v trénovacích datech nevyskytuje. V takovém případě tuto položku umístíme na konec lístku. Stejné pravidlo platí i pro položky, u kterých kategorie nebyla zvolena.

Abychom ověřili rychlost algoritmů, byla provedena dvě měření, jejichž výsledky jsou zobrazeny v tabulce 6.1. Prvním z nich je doba výpočtu modelu obchodu v závislosti na počtu kategorií v minulých nákupech. Její asymptotická časová složitost byla v návrhu odhadnuta jako kubická. Později jsme však změnili model na bigramový, což vedlo ke změně na kvadratickou složitost. Výsledné časy odpovídají spíše lineární složitosti, což není chybou, protože předpokládaná asymptotická složitost $O(c^2)$ nám udává horní omezení, neboli říká, že doba výpočtu by nikdy neměla být horší, než odhadnutá. Druhé měření bylo provedeno pro výpočet řadícího klíče z modelu obchodu. Ten je závislý na aktuálním počtu různých kategorií na nákupním lístku. Jeho asymptotická časová složitost pro nejhorší možný případ byla v návrhu odhadnuta jako kubická. Dle měření však můžeme vidět, že se blíží spíše kvadratické. Opět platí stejný princip horního omezení a výsledky tedy odpovídají návrhu.

Pokud to shrneme, inteligentní řazení je zajímavá funkce, která je mezi aplikacemi nákupních lístků unikátní. Uživatel sice musí pro její správnou funkčnost zaškrtávat položky přesně v tom pořadí, v jakém je dává do košíku, ale to není problém, který by způsoboval

Počet kategorií	Doba výpočtu modelu	Doba výpočtu klíče
2	3ms	12ms
4	6ms	19ms
8	8ms	42ms
16	14ms	140ms

Tabulka 6.1: Rychlost algoritmů inteligentního řazení

větší námahu. Uživatel na to pouze musí být upozorněn a s principem obeznámen. Řazení pak dosahuje dobrých výsledků shody s realitou. Pokud bychom se zamysleli, co do budoucna na řazení vylepšit, tak by to kromě optimalizace algoritmů bylo také znázornění důvěryhodnosti výsledků řazení. Není zde totiž vidět, s jakou pravděpodobností bylo dané pořadí kategorií vygenerováno a jak moc odpovídá modelu obchodu.

6.6 Vyhodnocení dat z databáze

MySQL databáze je primárně určena pro podporu webové aplikace na vzdálené přidávání nákupních lístků. Využití ovšem našla i pro verzi mobilní aplikace, která byla umístněna na Google Play market. Zde sloužila pro sběr dat, která uživatelé do aplikace zadají. Díky návrhu popsanému v sekci 3.15, můžeme nyní, kdy je aplikace již nějakou dobu na marketu, provést analýzu dat. Zejména nás budou zajímat položky a jejich četnost užívání, která je pro třicet nejvíce nakupovaných z nich zobrazena v tabulce 6.2. Můžeme tak vidět, že uživatelé více používají obecné názvy, než konkrétní značku nebo druh zboží. Většinou se jedná o potraviny každodenní potřeby, ale objevuje se zde i drogerie. Díky těmto datům může být databáze v budoucnu upravena. Například tím, že budou odstraněny položky, které nejsou používané. Dále je zde možnost přenesení počtu použití položek na globální úrovni do lokální databáze v mobilním zařízení. Nejvíce používané položky pak budou zobrazovány v našeptávači výše. S tím souvisí i možnost přímého propojení webové databáze se zařízením a stahování aktuální seznamu položek přímo ze serveru, o kterém by do budoucna mohlo být uvažováno.

Další informací, kterou můžeme z dat vyčíst, je chování uživatele při ovládání aplikace. Například v tabulce s nákupními lístky je mnoho řádků s názvy potravin. Z toho vyplývá, že část uživatelů nepochopila smysl spouštěcí aktivity sloužící k ukládání lístků a chybně do ní zadávají položky. Dále se zde také objevují záznamy, které obsahují více řádků. Při zadávání názvu nového lístku totiž není přiřazena znaku enter speciální funkčnost a při jeho stisku tak dochází k vytvoření znaku nového řádku, který je názvu lístku nežádoucí. V tabulce s položkami se pak objevují záznamy, v jejichž názvu je uvedena i cena. Někteří uživatelé by tak evidentně ocenili přidání dalšího atributu, do kterého by si mohli uložit orientační cenu zboží. Tyto a další implementační detaily tak mohou být lehce odhaleny a následně upraveny v další verzi aplikace.

6.7 Shrnutí

V této kapitole byly popsány jednotlivých části výsledné aplikace, tak jak byly implementovány v průběhu vývoje. Jejich výsledná podoba a funkčnost odpovídá až na několik drobných změn vytvořenému návrhu. Všechny části byly také úspěšně zakomponovány do jednoho celku, který je plně funkční. Uživatel naší aplikace tak může přijmout nákupní

Pořadí	Položky	Počet použití	Pořadí	Položky	Počet použití
1.	Mleko	226	16.	Maso	79
2.	Brambory	209	17.	Ovoce	75
3.	Rohliky	203	18.	Zelenina	71
4.	Cibule	160	19.	Sunka	70
5.	Maslo	157	20.	Rajcata	63
6.	Pecivo	147	21.	Salam	62
7.	Vejce	128	22.	Jar	61
8.	pivo	115	23.	Banany	52
9.	Olej	99	24.	Smetana	50
10.	Jogurty	96	25.	Kapesniky	50
11.	Jablka	88	26.	Kafe	49
12.	Syr	85	27.	Mouka Hladka	48
13.	Mrkev	84	28.	Housky	47
14.	Cukr	83	29.	Kecup	47
15.	Toaletni papir	82	30.	Zubni pasta	46

Tabulka 6.2: Počty použití jednotlivých položek v tabulce items na webovém serveru

lístek zaslaný z internetu a podle potřeby si ho upravit. Dále může vybrat obchod, ve kterém provede nákup, čímž dojde k seřazení položek na lístku dle rozmístění zboží v cílovém obchodě. Poté již může pohodlně provést nákup, přičemž si bude odškrtávat jednotlivé položky na lístku. Ušetří tak čas, který by musel investovat do starostí s papírovým nákupním lístkem.

Kapitola 7

Závěr

Zadáním této práce bylo vytvořit nějakým způsobem unikátní službu pro správu nákupních lístků, najít early adopters a s nimi testovat návrhy a hypotézy. Mobilní aplikace, která v rámci práce vznikla byla již od začátku řízena metodami vývoje, jež jsou aplikovány ve startupových projektech. Docházelo k pravidelnému testování předpokladů, shromažďování zpětné vazby, jejímu vyhodnocení a následnému určení dalšího směru vývoje. Tímto způsobem vznikla aplikace s unikátní metodou řazení položek na nákupním lístku, která je založena na předchozích nákupech uživatele v obchodě. Tuto funkci neobsahuje žádná ze známých konkurenčních aplikací. Mimo to bylo vytvořeno ještě webové rozhraní, přes které je možné do mobilního zařízení vzdáleně zasílat nové nákupní lístky.

Do budoucna bude aplikace určitě dále rozvíjena. Možností, o jaké funkce ji dále rozšířit je velké množství, ale primárně bude pozornost směřovat opět ke zpětné vazbě od early adopters, díky kterým tento projekt zaznamenal úspěch. Dále je možné se inspirovat u konkurence. Například načítáním zboží podle čárových kódů, zadáváním nových položek pomocí řeči nebo znovupoužíváním nákupních lístků. Z vlastních nápadů a postřehů by určitě bylo zajímavé implementovat vzdálené zasílání nákupních lístků z jednoho mobilního zařízení na druhé pomocí internetu.

Hlavním smyslem této práce ale bylo vytvořit software, který nebude pouze pro účel diplomové práce, ale budou ho využívat reální uživatelé, kterým ušetří čas a námahu. Toto se určitě splnit podařilo, protože v současné chvíli využívá aplikaci přes čtyři tisíce uživatelů po celém světě ke správě svých nákupních lístků. Aplikace navíc dosáhla po vystavení na Google Play rychlého růstu a dobrého hodnocení a na čas se díky tomu dostala na přední příčky mezi nejlepší aplikace v různých kategoriích. V současné době jsou téměř připraveny k nasazení nové funkce, u kterých se předpokládá, že posunou aplikaci mezi nejúspěšnější na marketu v dané kategorii.

Na diplomové práci si nejvíce cením zkušeností získaných při vývoji samotné aplikace a jejím prezentování na všech akcích a seminářích, kterých jsem se zúčastnil. V průběhu vypracovávání tohoto projektu, jsem se seznámil s technikami, které nejsou typické pro školní prostředí, ale používají se spíše pro klasické startupy. Potkal jsem také mnoho zajímavých lidí, kteří se v tomto prostředí pohybují. Zkušenosti vidím jako největší přínos, který pro mne tato práce měla.

Literatura

[1] Google. *Application Fundamentals* [online]. 2007- [cit. 2012-04-20]. Dostupné z: <http://developer.android.com/guide/topics/fundamentals.html>.

[2] Google. *Google Play* [online]. 2007- [cit. 2011-12-15]. Dostupné z: <https://play.google.com/store>.

[3] Google. *Persistent data* [online]. 2007- [cit. 2012-04-27]. Dostupné z: <http://developer.android.com/guide/topics/data/data-storage.html>.

[4] Google. *What is Android?* [online]. 2007- [cit. 2011-12-15]. Dostupné z: <http://developer.android.com/guide/basics/what-is-android.html>.

[5] Grudl, D., kolektiv vývojářů. *Dokumentace webového frameworku Nette* [online]. 2008- [cit. 2012-05-01]. Dostupné z: <http://doc.nette.org/cs/>.

[6] Jelinek, F. The development of an experimental discrete dictation recognizer. *Proceedings of the IEEE*. Nov. 1985, roč. 73, č. 11. S. 1616 – 1624. ISSN 0018-9219.

[7] Katz, S. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*. Mar 1987, roč. 35, č. 3. S. 400 – 401. ISSN 0096-3518.

[8] Kreibich, J. A. *Using SQLite*. [b.m.]: O'Reilly Media, 2010. ISBN 978-0596521189.

[9] Mahemoff, M. *HTML5 vs Native: The Mobile App Debate* [online]. 2011-06-03 [cit. 2012-04-15]. Dostupné z: <http://www.html5rocks.com/en/mobile/nativedebate/>.

[10] Maurya, A. *Running Lean: Iterate from Plan A to a Plan That Works*. [b.m.]: O'Reilly Media, 2012. ISBN ISBN 978-1449305178.

[11] Mednieks, Z., Dornin, L., Meike, G. B., Nakamura, M.. *Programming Android*. [b.m.]: O'Reilly Media, 2011. ISBN ISBN 978-1449389697.

[12] Murphy, M. L. *Android 2*. [b.m.]: Computer Press, 2011. ISBN ISBN 978-8025131947.

[13] Oracle. *Oracle Database Mobile Server Overview* [online]. [cit. 2012-04-27]. Dostupné z: <http://www.oracle.com/technetwork/products/database-mobile-server/overview/index.html>.

[14] Psutka, J. *Komunikace s pocitacem mluvenou reci.* [b.m.]: Academia, 1995. ISBN 8020002030.

[15] Ries, E. *The lean startup : how today's entrepreneurs use continuous innovation to create radically successful businesses.* New York: Crown Business, 2011. ISBN 9780307887894.

Dodatek A

Obsah CD

Seznam souborů na přiloženém cd včetně popisu:

- **xdoube01_dp.pdf** - tato diplomová práce ve formátu pdf
- **xdoube01_dp_zdroj.zip** - latexové zdrojové kódy této diplomové práce
- **xdoube01_ShoppingList_verze_1_0.zip** - zdrojové kódy mobilní aplikace v1.0
- **xdoube01_ShoppingList_verze_1_2.zip** - zdrojové kódy mobilní aplikace v1.2
- **xdoube01_ShoppingList.zip** - zdrojové kódy mobilní aplikace poslední verze
- **xdoube01_ShoppingList_web.zip** - zdrojové kódy webové aplikace
- **xdoube01_readme.txt** - návod a popis jednotlivých archivů