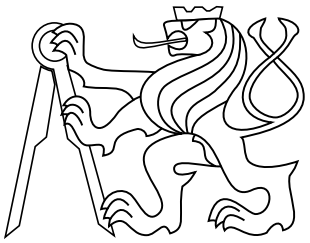
bakalářská práce

Software pro správu obsahu ledničky - Foodtracker

Ondřej Vaic



Únor 2017

Ing. Martin Balík, Ph.D.

České vysoké učení technické v Praze
Fakulta elektrotechnická, Katedra řídící techniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Vaic** Jméno: **Ondřej** Osobní číslo: **434978**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra řídicí techniky**

Studijní program: **Kybernetika a robotika**

Studijní obor: **Systémy a řízení**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Evidence potravin pro Android

Název bakalářské práce anglicky:

Food tracker for Android

Pokyny pro vypracování:

1. Navrhněte a implementujte aplikaci pro zařízení se systémem Android, která umožní sledování obsahu ledničky, trvanlivosti a množství potravin.
2. Implementujte možnost vkládání položek pomocí čárových kódů. Aplikace také umožní vytvoření nákupních seznamů a bude upozorňovat na blížící se expiraci potravin.
3. Aplikaci otestujte s uživateli a ověřte funkčnost i použitelnost (usability) uživatelského rozhraní.

Seznam doporučené literatury:

[1] Google and Open Handset Alliance n.d.: Android API Guide. http://developer.android.com/guide/index.html
[2] Lacko, L.: Vývoj aplikací pro Android. Brno: Computer Press, první vydání,2015, ISBN 978-80-251-4347-6.
[3] Watson, G.: ORMLite Package. Version 5.0, July 2016. http://ormlite.com/docs/ormlite.pdf

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Martin Balík, Ph.D., Centrum znalostního managementu FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **14.02.2017** Termín odevzdání bakalářské práce: **25.05.2018**

Platnost zadání bakalářské práce: **30.09.2018**

Ing. Martin Balík, Ph.D.	prof. Ing. Michael Šebek, DrSc.	prof. Ing. Pavel Ripka, CSc.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. | Datum převzetí zadání

Podpis studenta

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Poděkování

Chtěl bych poděkovat mému vedoucímu, Martinu Balíkovi, za trpělivost, jeho rady a předání zkušeností v průběhu vytváření této práce. Dále bych také chtěl poděkovat celé komunitě okolo Androidu za možnost učit se z jejich poznatků a pracovat s některými jejich technologiemi.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Abstrakt

Smyslem této práce je vyvinout a zdokumentovat průběh vývoje mobilní aplikace, která si klade za úkol zjednodušení práce s potravinami a domácími potřebami. Uživateli budou nabídnuty funkce k práci s obsahem virtuální lednice a na práci s tvorbou a prohlížení nákupních seznamů. Aplikace je mířená na platformu Android a napsaná v jazyce Java. V průběhu kapitol budou chronologicky ukázány jednotlivé segmenty esenciální pro vytvoření aplikace. Mezi tyto části patří analýza zadaného problému, návrh aplikace a samotný vývoj zakončený testováním.

Klíčová slova

Android,vývoj,software,mobilní,aplikace,lednice,nakupování,jídlo,databáze,čárkové kódy

Abstrakt

The purpose of this thesis is to create and describe the development of mobile application, which strives to simplify working with groceries and other household products. The user will gain access to tools for managing a virtual fridge and for creating and reviewing of shopping lists. The application is aimed for the Android platform and is written in Java language. Throughout the chapters will chronologicaly be uncovered the essential sections for such process. Among these sections is analysis of the given problem, design of the application and then the development itself, finished by testing.

Keywords

Android,software,development,mobile,aplications,fridge,shopping,food,database,barcode

Obsah

Zkratky

Použité zkratky v následujícím textu...

SW	software
FT	FoodTracker
GUI	Graphical User Interface - grafické uživatelské rozhraní
UI	User Interface - uživatelské rozhraní
UX	User Experience - uživatelský zážitek
DI	Dependency Injection - injikování závislostí
API	Application Programming Interface - programovací rozhraní aplikace
DAO	Data access object - objektový přístup práce s databází
ID	Identification - identifikační číslo
SDK	Software Development Kit - balíček nástrojů pro vývoj
PC	Personal Computer - počítač
POJO	Plain Old Java Object
JPA	Java persistence API

1 Úvod

1.1 Motivace

V průběhu mého studia na Fakultě elektrotechnické jsem nalezl zábavu v programování a potažmo ve výrobě softwaru a jako zajímavý vstupní bod mi připadaly mobilní aplikace. Jedním z hlavních důvodů je, že mobilní aplikace jsou trendem tohoto desetiletí, hlavně díky rychlosti rozvoje v oblasti mobilních telefonů, který jde ruku v ruce s přístupností těchto zařízení velké části populace[1], v technologicky rozvinutých zemích se pak jedná o většinu obyvatel[2]. Díky této přístupnosti je možné oslovit potenciálně velkou cílovou skupinu i jednoduchou aplikací. Další velmi kladnou stránkou je, že na vývoji mobilních aplikací se často nepodílejí velké softwarové týmy, ale k vytvoření použitelného produktu stačí determinovaný jedinec.
Kombinací předchozích bodů jsem dospěl k závěru, že by se jednalo o dobré téma bakalářské práce, kde bych navíc pod vedením zkušeného programátora získal mnohem víc než jen při osobním experimentování a rozhodl se o procesu vývoje napsat tuto práci, která jedince s podobným úmyslem může pomoct nasměrovat při překonávání podobných problémů.

1.2 Cíle práce

Cílem této práce je podat zprávu o celém průběhu návrhu a vývoje softwaru, v našem případě se jedná o Android aplikaci, nicméně velmi podobný postup by se dal aplikovat na jakékoliv jiné odvětví softwarového vývoje. Jak bylo ale zmíněno výše, na vytvoření aplikace stačí sice jeden člověk, na druhou stranu ale musí zastat spektrum různých funkcí, které jsou v produkci softwaru zastávány často i vyučenými odborníky pro danou tématiku. Velkou výzvu například vidím ve vytváření grafického rozhraní a budování kvalitního uživatelského zážitku, ale také v architektuře celé aplikace, kde se dá jednoduše doplatit na nedostatek zkušeností a podcenění složitosti celého problému. Ve vytváření této aplikace nám ale hodně pomáhají sofistikované nástroje vytvořené komunitou okolo Androidu, tedy není cílem znovu objevovat koncept kola, ale využít dostupné nástroje a pomocí nich dodat uživateli smyslplnou a užitečnou aplikaci.
V našem případě se bude jednat o aplikaci na správu obsahu lednice a vytváření nákupních seznamů. Tato myšlenka je dostatečně jednoduchá, aby se dala pojmout jako aplikace, nicméně pořád nabízí kvalitní základ pro implementaci velké škály funkčnosti a samotné propojení dvou hlavních modulů nabízí zajímavé funkce které by cílový uživatel mohl ocenit.

1.3 Osnova práce

Po krátkém úvodu v první kapitole do dané problematiky se dostáváme do druhé kapitoly, ve které jsou rozebrány znalosti. U znalostí se budeme držet pouze minimálního potřebného množství k pochopení textu práce , na více informací pak bude odkázáno směrem k internetovým publikacím, kde se Android honosí velmi kvalitní dokumentací.

Následuje kapitola třetí, která je zaměřená na rozbor. Tady analyzujeme potřebné funkce a situaci na trhu s podobnými aplikacemi. Důležitými informacemi z této kapitoly budou odpovědi na otázky jako:„Jsme schopni zaplnit mezeru na námi cíleném trhu?", popřípadě: „Jsme alespoň schopni konkurovat největším značkám které jsou již na trhu ustáleny?". Toto porovnání vyplývá hlavně z pohledu na námi nabízené funkce a schopnost navrhnout kvalitní grafické rozhraní a následného pohledu na tyto elementy v konkurenčních aplikacích.

Odtud se pak přesouváme do kapitoly čtvrté, která pojednává o procesu návrhu celé mobilní aplikace a způsobu implementace klíčových vlastností a funkcí vytvořené aplikace. V této kapitole jsou také rozebrány úskalí která byla během návrhu objevena a způsoby jakým byly dané problémy vyřešeny. Tím se dostáváme do kapitoly páté, kde jsou zobrazeny hlavně ukázky z již hotové aplikace s vysvětlivkami k funkci jednotlivých modulů, či složitějším částem kódu. Také je rozebrána výsledná architektura celé aplikace a popřípadě vysvětlení způsobu přístupu a návrhových vzorů. Také jsou představeny použité technologie a software potřebný k vytvoření námi žádané aplikace. Uvádí se tu jaké technologie byly použity ve výsledné verzi, tak alternativní technologie, které také mohly zastávat námi požadovanou funkčnost. U některých technologií se podíváme i na porovnání a důvody pro vybrání té které technologie.

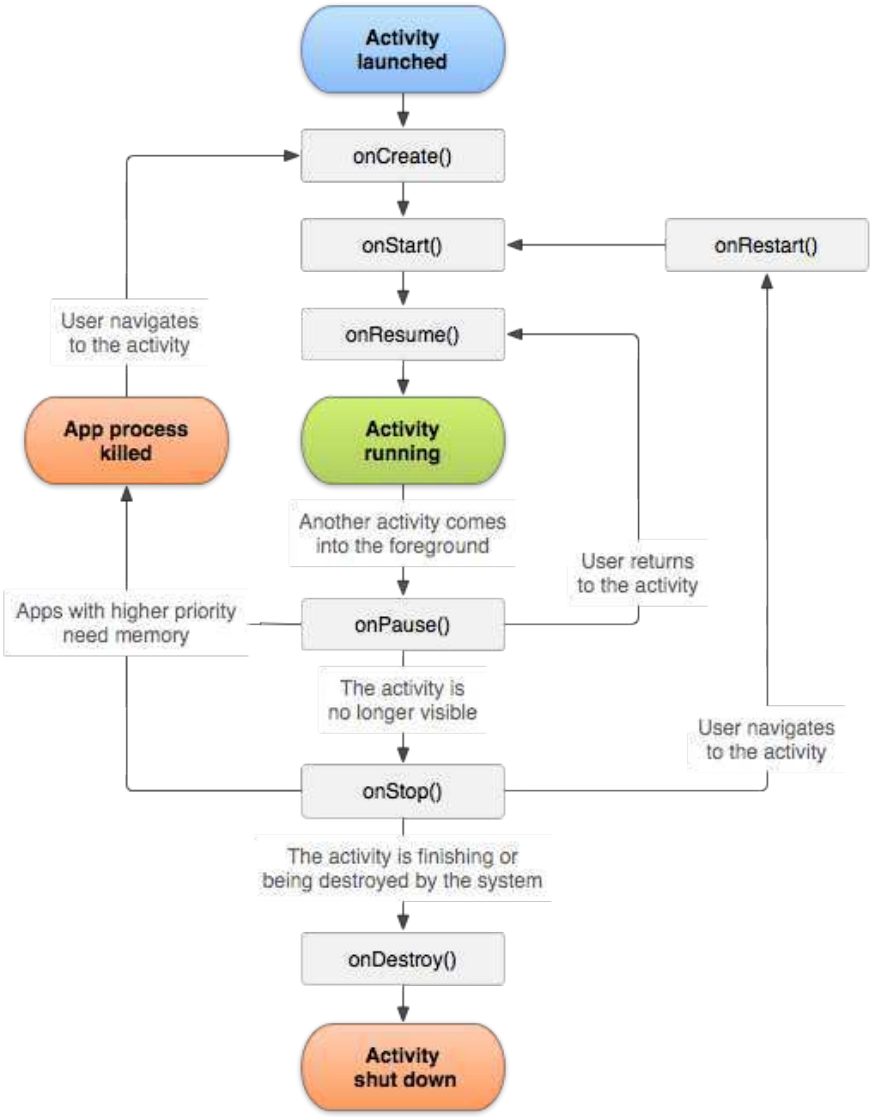
V kapitole šesté je pak hlavní součástí testování, přesněji řečeno uživatelské testování, nebo-li objektivní kritika od několika subjektů, kteří tuto aplikaci krátkodobě používali a jak tyto návrhy ovlivnily výsledný vzhled vyvíjeného produktu. V závěru je poté popsán průběh zvěřejnění na Google trhu aplikací, zhodnocen průběh, výsledek práce a jsou rozebrána další možná vylepšení.

2 Prerekvizity

Pro úspěšný vývoj Android aplikace jsou potřeba určité znalosti o kterých se bude mluvit v této kapitole. Součástí jsou pouze termíny související s vývojem aplikace na Android a cílem této kapitoly je čtenáře s těmito termíny seznámit.

2.1 Aktivity

Základní stavební kámen celého grafického rozhraní je aktivita[4]. Velmi zjednodušeně řečeno, aktivita je programová reprezentace obrazovky, kterou uživatel může otevřít. Ne



Obrázek 1 Životní cyklus aktivity.[3]

však všechny obrazovky jsou nutně aktivity, takovýmto zobrazením se říká fragmenty a k nim se dostaneme v další částí, kde budou vysvětleny i rozdíly. Každá aplikace která využívá grafické rozhraní a neběží pouze na pozadí, potřebuje alespoň jednu aktivitu a v ní jsou poté začleněny další grafické prvky jako tlačítka, textové zobrazení a podobně.

Důležité pro práci s aktivitami je porozumět jejich životnímu cyklu. Pomocí přepisování těchto jednotlivých funkcí má vývojář větší kontrolu nad tím jak aplikace reaguje. Například místo toho aby se všechny proměnné celé aplikace zbytečně inicializovaly při spuštění aplikace, můžeme přepsat methody onCreate() a jen při samotném otevření aktivity provést potřebné kroky. Jak celý životní cyklus vypadá je zobrazeno na obrázku 1

Esenciální pro práci s aplikacemi je také pochopení termínu Context, který se váže jak s aktivitami, tak s celou aplikací. Kontext[5] je rozhraní které podává přístup k prostředkům a třídám které jsou specifické pro naší aplikaci. Je také důležitý při spouštění aktivit. Je potřeba i pro inicializaci některých prvků které přímo nesouvisí s grafickým rozhraním, jako je třeba pomocník pro práci s databází.

2.2 Fragmenty

Jak bylo řečeno výše, fragmenty [6] jsou další z možností jak zobrazit uživatelské rozhraní uživateli, s tím rozdílem že fragment by měl reprezentovat část logického celku uživatelského rozhraní. Není to však jediný rozdíl, mezi další rozdíly patří

- Fragment nemůže existovat nezávisle, vždy musí být spojen s nějakou aktivitou. Z toho vyplývá potřeba alespoň pro jednu aktivitu v aplikaci, jak bylo zmíněno v 2.1.
- Fragment může být znovu použit v různých aktivitách.
- Do fragmentu se nelze vrátit pomocí zásobníku posledních aktivit.

Fragmenty jsou ve výsledku mnohem lepší k zobrazování informací, hlavně z důvodu znovupoužitelnosti a plně jsem je bohužel dokázal ocenit až při dokončování finální aplikace.

2.3 Ostatní grafické elementy

Zbývají nám elementy které si nezaslouží jejich vlastní kapitolu, patří sem elementy díky nimž dosahujeme responzivnosti na uživatelské akce. Jedná se o **dialogy** a **toasty**. Dialog je způsob zobrazení krátké zprávy uživateli a zobrazení nějakých možností jak na tuto zprávu reagovat. V naší aplikaci jsou třeba dialogy využity jako potvrzení v případě mazání z databáze, nebo jako zobrazení detailu položky lednice.
Druhý element, toast, se používá k zobrazení krátké zprávy, s tím rozdílem že nenabízí uživateli reakci jako dialogy. Jsou v naší aplikaci využívány například při uložení položky do databáze. Poté co jsou vyplněny všechny potřebné informace a je zmáčknuto tlačítko uložit produkt, by se mohlo zdát že zařízení stisk tlačítka nezaregistrovalo, proto je zobrazena krátká zpráva na spodu obrazovky která uživateli sdělí že tato akce proběhla úspěšně. Toasty jsou využívány na víc místech, ale princip zůstává stejný. V profesionální aplikaci by bylo příjemnější vytvořit pro uživatele nějakou animaci která dává najevo stejnou informaci, ale míň intrusivním způsobem.

2.4 Services

Services[7] jsou procesy, které běží bez vědomí uživatele na pozadí a zpracovávájí nějaké větší množství dat, nebo se například dají použít jako v našem případě k posílání notifikací. Tuto notifikaci posíláme i když uživatel zrovna aplikaci nepoužívá, ale děje se něco o čem by měl být upozorněn. Service sami o sobě neposkytují žádné uživatelské rozhraní a proto se kombinují s notifikacemi, popřípadě se používají na složitější nízkoúrovňové operace jako je komunikace po síti nebo se vstupními/výstupními zařízeními nebo mohou třeba přehrávat hudbu.

2.5 Intent

Poslední důležitou znalostí pro práci s dříve vyjmenovanými komponenty je Intent. Intent se používá jako zpráva do další komponenty, pomocí ní můžeme spouštět jiné aktivity nebo services a dodávat do těchto komponent data. Nejedná se o složitý koncept jak je vidět z následujícího příkladu

```
void moveToAddingFood() {
        Intent intent = new Intent(MainActivity.this, AddFoodActivity.class);
        intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        startActivity(intent);
    }
```

K vytvoření je zapotřebí pouze současný a cílový kontext a poté nastavit o jaký intent se vlastně jedná, pomocí flagu. O spuštění aktivity se pak postará operační systém.

3 Analýza

Jak bylo naznačeno v osnově práce, cílem této kapitoly je zhodnotit konkurenci v námi cílené oblasti trhu a na základě tohoto zhodnocení rozebrat námi poskytované funkce a softwarovou analýzu. Je nadmíru důležité tento krok provést jako první, pokud je naším cílem soutěžit na trhu, ale i pokud je naším cílem vydávat aplikace pouze amatérsky, neuškodí se inspirovat prvky úspěšných aplikací, popřípadě se poučit z těch méně vydařených. Předmětem zájmu naší analýzy bude hlavně spektrum poskytovaných funkcí, podíváme se ale i na další informace jako je například počet stažení. Tyhle informace nám můžou dále pomoct při zjišťování zájmu o daný typ aplikace, popřípadě není-li na trhu místo připravené přímo pro nás.

O samotné analýze by se nejspíš dala napsat samostatná práce, nicméně jako součást naší práce se omezíme pouze na rychlý průzkum konkurečních aplikací, tedy těch které se svojí funkčností překrývají s naším FoodTrackerem. Tím zjistíme, co vlastně můžeme naším zákazníkům nabídnout, popřípadě jestli má projekt vůbec budoucnost. Dále je nutno poznamenat že není možno a ani není cílem této kapitoly popsat všechny aplikace, které nám na tomto obrovském trhu konkurují. Proto se omezím na výběr několika aplikací, které bych osobně jako uživatel chtěl používat a na druhou stranu pak uvedu i ty aplikace které mají nějaké nedostatky ze kterých by se naše aplikace mohla poučit. Tato sekce je nadále rozdělena na konkurenty v oblasti nákupních seznamů a oblasti sledování obsahu ledničky, protože jsou to naše stěžejní funkce FoodTrackeru a na trhu momentálně neexistuje aplikace s podobným nápadem propojení těchto dvou funkcí.

3.1 Analýza trhu

3.1.1 Nákupní seznamy

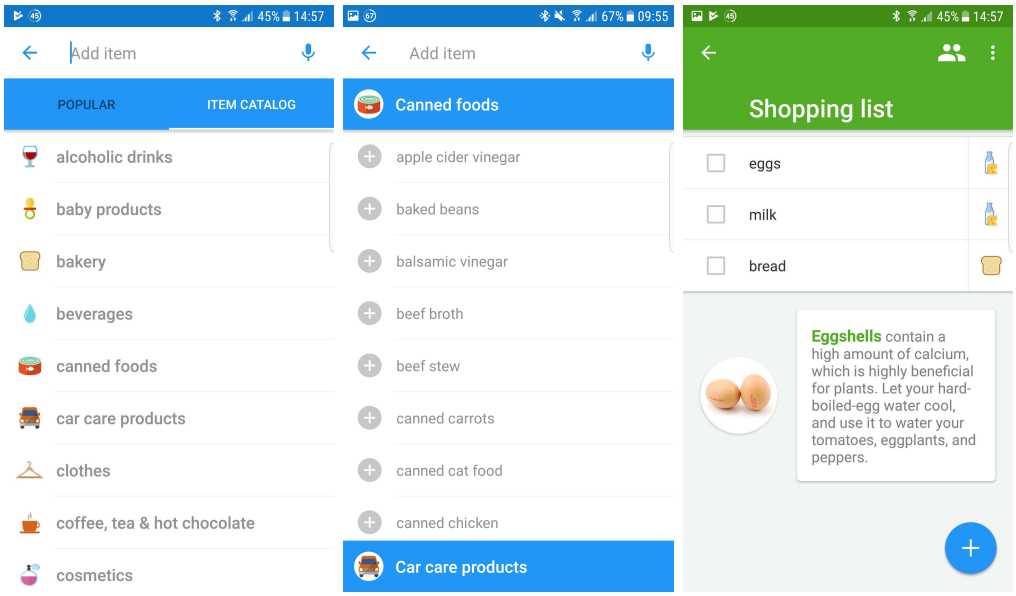
Listonic

Listonic[1] je aplikace na vytváření nákupních seznamů, která již delší dobu dominuje trh se svými 7 miliony stažení. Není se čemu divit, pro uživatele je připraven příjemný zážitek v podobě povedeného grafického rozhraní a svižného fungování. Tohoto pocitu je docíleno hlavně animacemi jako jsou přechody mezi aktivitami a obrovské nabídky funkcí která velmi zjednodušuje samotné vytváření seznamů. Mezi tyto funkce patří například :

- Ovládání hlasem přes Google Voice
- Návrhy populárních produktů mezi ostatními uživateli
- Rozřazení položek do katalogu, což jsou kategorie ve kterých můžete okamžitě najít hledaný produkt
- Automatické doplňování textu při vyhledávání(tím se myslí že nabízí smysluplné možnosti, tedy hlavně potraviny a domácí potřeby)
- Propojení s Google účtem, což umožňuje například sdílení nákupního seznamu

[1] *Chytrý nákupní seznam Listonic.* URL: `https://play.google.com/store/apps/details?id=com.l&hl=cs` (cit. 10. 05. 2018).

A také je určitě potřeba zmínit že jsou hotové překlady pro několik jazyků. Jediný záporný bod vidím v tom, že jsou v aplikaci přítomny reklamy. Nejsou nijak zvlášť intruzivní ale i tak to ničí jinak kvalitní dojem.



a) Katalog všech kategorií **b)** Otevřená kategorie **c)** Nákupní seznam

Obrázek 2 Ukázka grafického rozhraní aplikace Listonic.

OutOfMilk

OutOfMilk[2] je také konkurentem naší aplikaci v oblasti vytváření nákupních seznamů, nicméně je v této oblasti více než kvalifikovaný. Další oblasti do kterých se odvíjí zbytek funkčnosti nám sice nekonkurují, ovšem množství možností by mohlo být pro případného zákazníka prodejním argumentem, jedná se například o To-Do list, seznam receptů, lokální slevy (určitě nefunkční v České republice, podle internetového průzkumu je funkční v USA), a takzvaný pantry list, což umožňuje uživateli kontrolovat zásoby ledničky, ale ne s takovou obratností s jakou plánujeme FoodTracker.

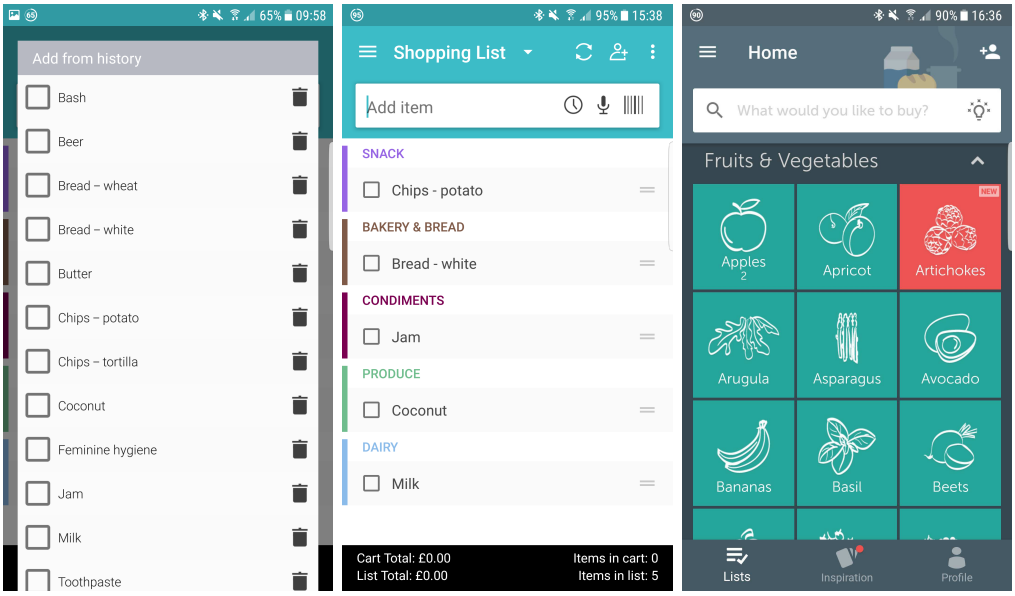
Spektrum funkcí které nabízí se s pár výjimkami velmi podobá Listonicu. Tou výjimkou je načítání nákupu přes čárkové kódy, jinak opět nacházíme funkce jako je implementace Google Voice, Google účtů a autocorrect specifikovaný přímo na potraviny. Jako jedinou výhodu oproti Listonicu vidím fakt, že nejsou přítomny reklamy a možnost načítání přes čárkové kódy. Na druhou stranu grafické rozhraní není tak příjemné. I přes to že je očividně kladen důraz na jednoduchost, je na můj vkus použito příliš mnoho základních elementů které jsou k dispozici v Android SDK a aplikace tak ve výsledku působí graficky nedodělaně.

Bring

Bring[3] je poslední kvalitní konkurent v oblasti vytváření nákupních seznamů, což potvrzuje i 1 milión stažení na obchodu Google Play. Co je zajímavé na této aplikaci

[2] *Out of Milk - Grocery Shopping List.* URL: `https://play.google.com/store/apps/details?id=com.capigami.outofmilk` (cit. 10. 05. 2018).

[3] *Bring! Grocery Shopping List.* URL: `https://play.google.com/store/apps/details?id=ch.publisheria.bring` (cit. 10. 05. 2018).



a) Přidávání z historie **b)** Nákupní seznam **c)** Katalog kategorií

Obrázek 3 První dva obrázky zleva jsou z aplikace OutOfMilk,poslední je z Bring.

je jeho prvotřídně navržené grafické rozhraní, které obsahuje nabídku všech možných produktů provedenou v metro stylu. Samozřejmě nechybí ani vyhledávání pomocí textového zadávání, ale grafické vyhledávání bude u zákazníků jasným favoritem. Dále je také zajímavá funkce nazvaná “inspirace”, která obsahuje oblíbené recepty a k nim rovnou vytvořené nákupní seznamy které vám umožní jednoduše nakoupit potřebné ingredience. Umožnuje také propojení přes Google účet, ale bohužel neobsahuje pokročilé funkce přechozích aplikací jako je Google Voice, nebo čtení čárkových kódů. Očividně je tu opět kladen důraz na co nejjednodušší aplikaci, která nemá mnoho zbytečných funkcí, které ve výsledku běžný uživatel stejně nepoužije, ale spíš je vyvíjena snaha nabídnout uživateli prostředí, ve kterém může co nejrychleji vytvořit nákupní seznam a poté se v něm jednoduše orientovat, což je výborně provedeno.

Srovnání

V průběhu této podkapitoly se objevilo spoustu pojmů a výsledek by byl složitý si představit, proto jsou hlavní body vytknuty v tabulce 1.

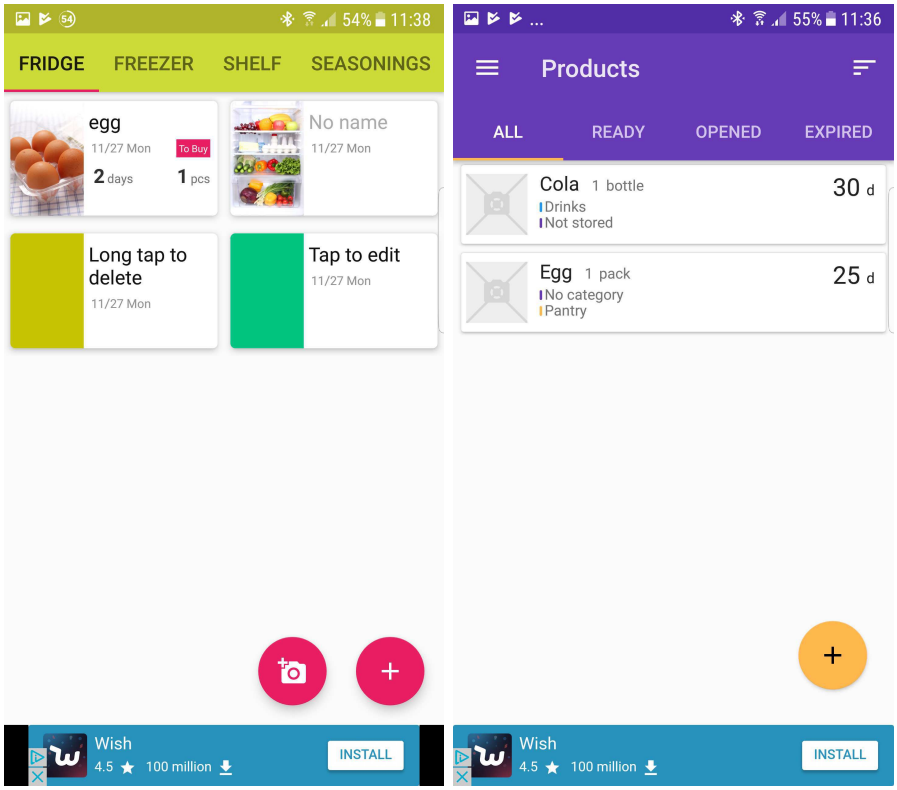
Tabulka 1 Srovnání aplikací na nákupní seznamy.

Aplikace	**Listonic**	**OutOfMilk**	**Bring**
GUI	4.5*	4*	5*
Rozdělení do kategorií	Ano	Ano	Ano
Google účet	Ano	Ano	Ano
Google Voice	Ano	Ano	Ne
Reklamy	Ano	Ano	Ne
Čtení čárkových kódů	Ne	Ano	Ne
Ostatní		To-Do list recepty lokální slevy	Inspirace na recepty

Výsledek je tedy takový, že v oblasti nákupních seznamů je již několik usazených titánů a dostat se mezi ně je bez větší inovace prakticky nemožné. Než začneme kreslit závěry, podívejme se nejdříve do oblasti sledování lednice.

3.1.2 Sledování obsahu ledničky

V oblasti sledování ledničky je mnohonásobně menší konkurence a to jak kvalitativně tak kvantitativně. Podařilo se mi najít pouze 2 aplikace s větším množstvím funkcí, obě se ale pohybují okolo 10 000 stažení a na průměrném hodnocení. Jedná se o MyKura[4] a BestBefore[5]. Nejvíc tyto aplikace strádají v oblasti GUI, ukládání položek je dost těžkopádné a zobrazování uložených potravin neestetické. Obsahují mimo jiné i funkce, které jsou pro uživatele ve výsledku zbytečné. Z mého pohledu není třeba rozlišovat zda mám produkt uložen v lednici, či v mrazáku, jak vidíme např. na obrázku 4a, důležité ale je rychle najít které produkty brzy expirují a kterých mám nedostatek. Daným rozdělením akorát zavádíme do aplikace chaos. Zobrazení uložených produktů pomocí karet je také z mého pohledu zbytečné, akorát zabíráme místo na kterém jsme mohli efektivně zobrazit potřebné informace.



a) MyKura **b)** BestBefore

Obrázek 4 Grafické rozhraní aplikací na sledování zásob.

Ani k jejich funkčnosti není moc co dodat, obsahují sice také čítač čárových kódů, ale implementace je špatně provedená, protože pokud váš čárový kód není nalezen, jste donucen načítat vstupy textově, což se dá obejít přiřazením čárového kódu k nějakému

[4] *Fridge, Foods, Expiration date.* URL: `https://play.google.com/store/apps/details?id=jp.gr.java_conf.indoorcorgi.mykura` (cit. 10. 05. 2018).

[5] *Best Before - Food Tracker.* URL: `https://play.google.com/store/apps/details?id=com.peytu.bestbefore` (cit. 10. 05. 2018).

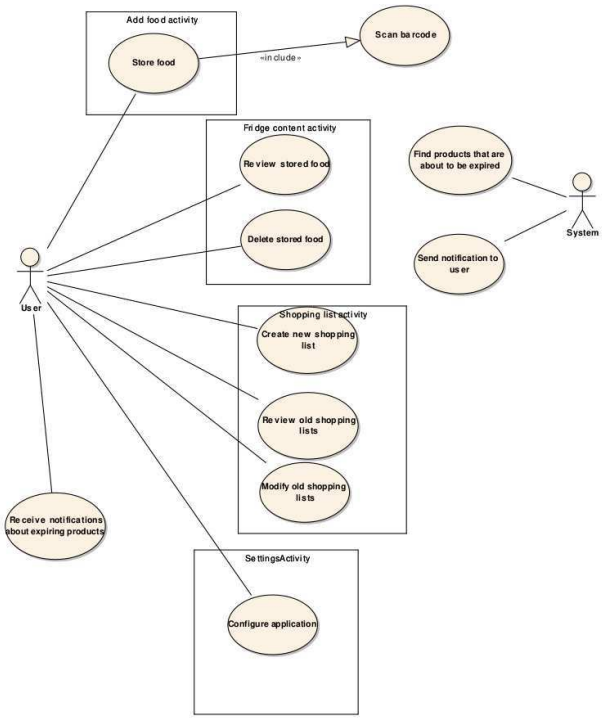
produktu v databázi. Tím se problém přesune jen pro první načítání pro každý produkt a každé další načítání je otázka pár sekund.

3.1.3 Závěr analýzy konkurence

Jak je vidno z analýzy, v oblasti správy obsahu lednice by FoodTracker mohl doopravdy zazářit, a jak vyplynulo i z mého neoficiálního průzkumu provedeného v okruhu přátel, lidé s vlastní domácností o aplikaci na spravování lednice opravdu stojí. Nízký počet stažení pro poslední dvě aplikace nemusí nutně vyplývat z nezájmu trhu, ale z absence kvalitního kandidáta. Pokud by se nám podařilo získat uživatelskou základnu díky našemu spravování lednice, neměl by být problém se poté usadit i v oblasti nákupních seznamů

Na konec by bylo dobré si z této analýzy odnést i nějaké ponaučení z těch méně úspěšných kandidátů. Jako základní kámen úspěchu je samozřejmě kvalitně zvládnutý backend aplikace, ovšem ten měly všechny výše uvedené aplikace a i přes to bych některé z nich odmítal používat. Naším cílem je tedy i bez přednostních zkušeností s tvorbou UI a UX vytvořit co nejpříjemnější uživatelské rozhraní a ve fázi testování opravdu zjistit co se uživatelům líbí a nelíbí a pokusit se na jejich názory nějak reflektovat. Nad rámec tohoto projektu by bylo dodat animace pro veškeré přechody v UI a třeba dodat i vlastní jednoduchou grafiku. Více k tomuto tématu bude v závěrečné kapitole.

3.2 Softwarová analýza



Obrázek 5 Use case diagram FoodTrackeru.

3.2.1 Specifické požadavky

FoodTracker je mobilní aplikace, jejíž cílem je zjednodušit práci s potravinami v domácnosti a to pomocí funkcí správy lednice a vytváření nákupních seznamů. Stav, kvalita a funkčnost výsledného software je definována pomocí specifických požadavků, které si v této podkapitole rozebereme. Tyto požadavky budou dále rozděleny na funkcionální a nefunkcionální. Mezi funkcionální patří výpis funkcí podle požadavků zadavatele, tyto funkce jsou zobrazeny na user diagramu na obrázku 5. Funkce jsou rozdělené podle jednotlivých modulů, u kterých probereme do hloubky funkčnost dále v této kapitole. Nefunkcionální požadavky kladou důraz na výsledné kvality aplikace a mezi ně patří

- Android aplikace - výsledná aplikace bude funkční na platformě Android, který má Android API alespoň 15.
- Spolehlivost - vyvarovat se chybám při práci s databází a pokud možno nikdy neporušit datovou strukturu a validitu dat, tak aby aplikaci nehrozilo padání a aby zobrazovaná data nebyla v rozporu s tím co zadal uživatel.
- Přístupnost - poskytnout uživateli nástroj, který v práci s položkami pomáhá a k jeho používání není potřeba žádná speciální znalost ani schopnosti.
- Udržitelnost - navrhnout celou aplikaci tak, aby se dala kontinuálně používat i pro větší množství dat.
- Výkon - pracovat s daty takovým způsobem, která zbytečně nevytěžují hardware telefonu.
- Znovupoužitelnost - generalizovat přístup ke všem domácím potřebám, aby se aplikace obecně dala použít ke všem možným variantám zakoupitelných produktů.

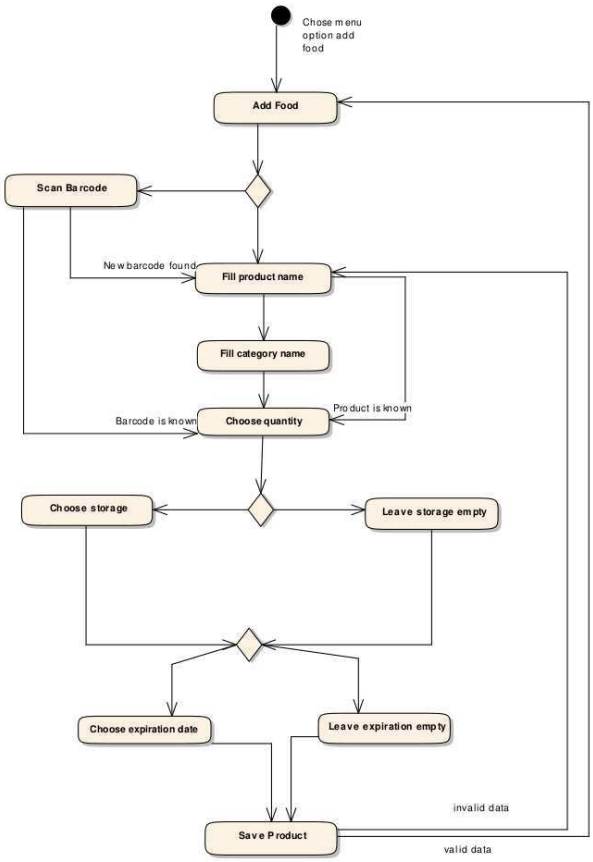
Tyto kvality nám mimo jiné zajišťují, že v případě nutnosti by bylo možné aplikaci rozšířit o další funkce, ať už se jedná o propojení s paralelně vyvíjenou aplikací na vytváření receptů, které můžeme například poskytnout současný stav lednice, nebo chytré vytváření nákupních seznamů, které bude vytvářet rychlé nákupní seznamy podle četnosti spotřebovávání položek. Nyní se podívejme na jednotlivé moduly a analýzu jejich implementace.

3.2.2 Analýza implementace procesů

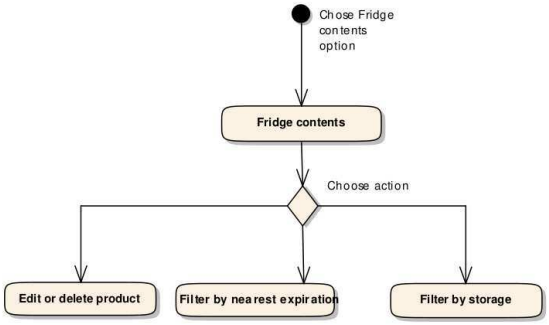
Načítání položek je aktivita, ve které uživatel ukládá své produkty do databáze. Hlavním cílem tohoto modulu je co nejvíce zjednodušit proces načítání, měl by tedy obsahovat funkce které předvyplňují co nejvíce informací z dříve dostupných dat. Proto je dostupné i načítání čárových kódů. Proces načítání je zobrazen na aktivity diagramu na obrázku 6.

Správa zásob by měla uživateli nabídnout všechny nástroje, které může potřebovat při hledání nějak zajímavých položek, například těch které brzo expirují. Vyfiltrovat ty které se nachází doma a nebo jen ty které má na chatě. Také je potřeba mazat zastaralé položky a měnit kvantitu těch které jsme již částečně spotřebovali. Do budoucna by se určitě hodilo i začít sbírat informace na frekvenci používání produktů, pomocí které by jsme mohli signalizovat položky, které brzy dojdou a například nabídnout rychle vytvořený nákupní seznam s potřebnými položkami.

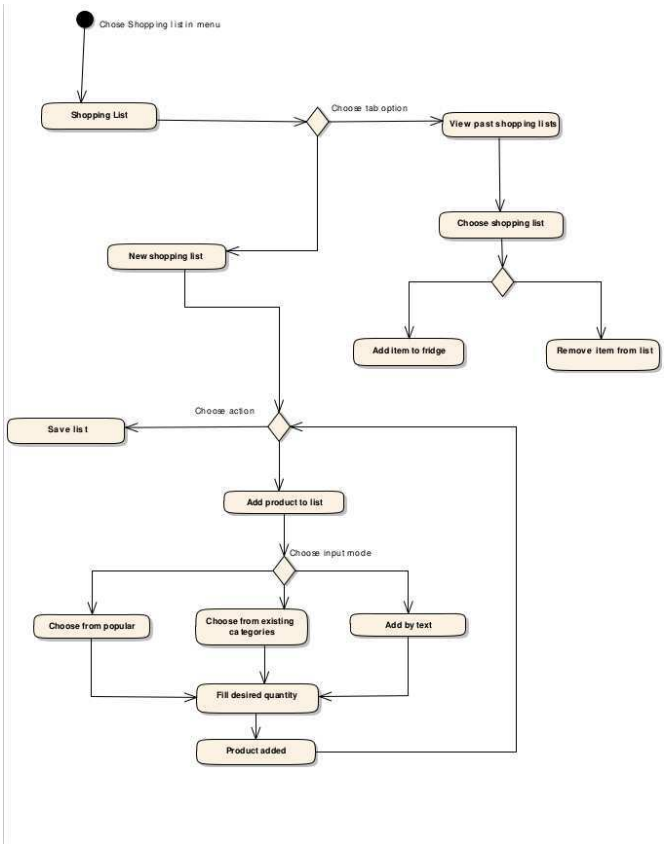
Vytváření a prohlížení seznamů jsou zobrazeny pouze na jednom diagramu na obrázku 8, kvůli prolínání jejich funkcí. Cílem je opět co nejvíce zjednodušit vytváření seznamů. Uživateli bude tedy nabídnuta možnost přidávat položky podle kategorií, které si sám při ukládání může vytvořit. Nesmí chybět ani možnost přidávání pomocí čistého textu pro případy nových produktů, které zatím nejsou v databázi. Také mi přišlo výhodné



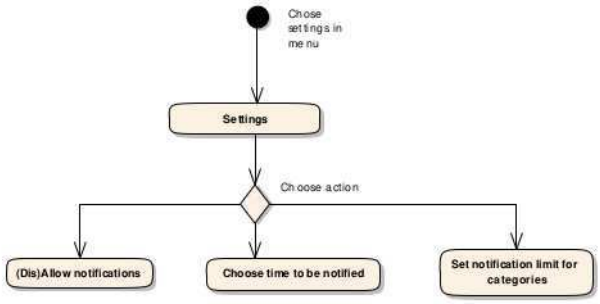
Obrázek 6 Aktivity diagram načítání položek.



Obrázek 7 Aktivity diagram zobrazovaní zásob.



Obrázek 8 Aktivity diagram vytváření a procházení nákupních seznamů.



Obrázek 9 Aktivity diagram tvorby konfigurací.

vytvořit kategorii populárních předmětů, kde uživatel bude moci najít 10 položek které přidává do nákupních seznamů nejčastěji.

Poslední z uživatelsky kontrolovaných funkcí je konfigurace notifikací. Důležité v této sekci zejména je, mít možnost změnit limit ve kterém chce být uživatel notifikován o expirujících produktech specifických kategorií. Má tak například možnost rozlišit kategorii mléčných produktů, o kterých bude dostávat notifikace pár dní předem, oproti těstovinám, kde bude notifikace stačit třeba měsíc dopředu. Také je potřeba umožnit uživateli notifikace vypínat a zapínat, v případě že o danou funkčnost nestojí. Poslední funkcí je vybírání času doručení notifikací, aby uživateli nechodili zprávy v půlce noci.

3.2.3 Shromažďovaná data

Ke správnému fungování je potřeba využití databáze. Mezi shromažďovaná data ve finální verzi patří

- Kategorie produktů
- Produkty a jejich jednotky
- Značky zakoupených produktů a jejich čárové kódy
- Historie nákupních seznamů a preferencí
- Konfigurační data

Více k jednotlivým kategoriím a k nim potřebným datům je v kapitole 4.1.

3.3 Funkce inspirované z analýzy konkurence

Při procházení konkurenčních aplikací jsem narazil na několik zajímavých funkcí, které by jsem rád implementoval do FoodTrackeru. V první řadě se jako velmi efektivní jevila funkce Google Voice. Tato technologie je v dnešní době již velmi pokročilá a při vlastním testování se mi v minimum případů stávalo, že by mi software špatně rozuměl. Dále se mi jako naprosto esenciální zdá možnost rozřazení produktů do jednotlivých kategorií. Jednak to velmi usnadňuje proces načítání a pomáhá to při notifikaci expirujících potravin, například můžeme přidat možnost nastavení časového limitu ve kterém chce být uživatel notifikován o blížící se expiraci. Byl by naprostý nesmysl tyto notifikace posílat unifikovaně pro veškeré kategorie.

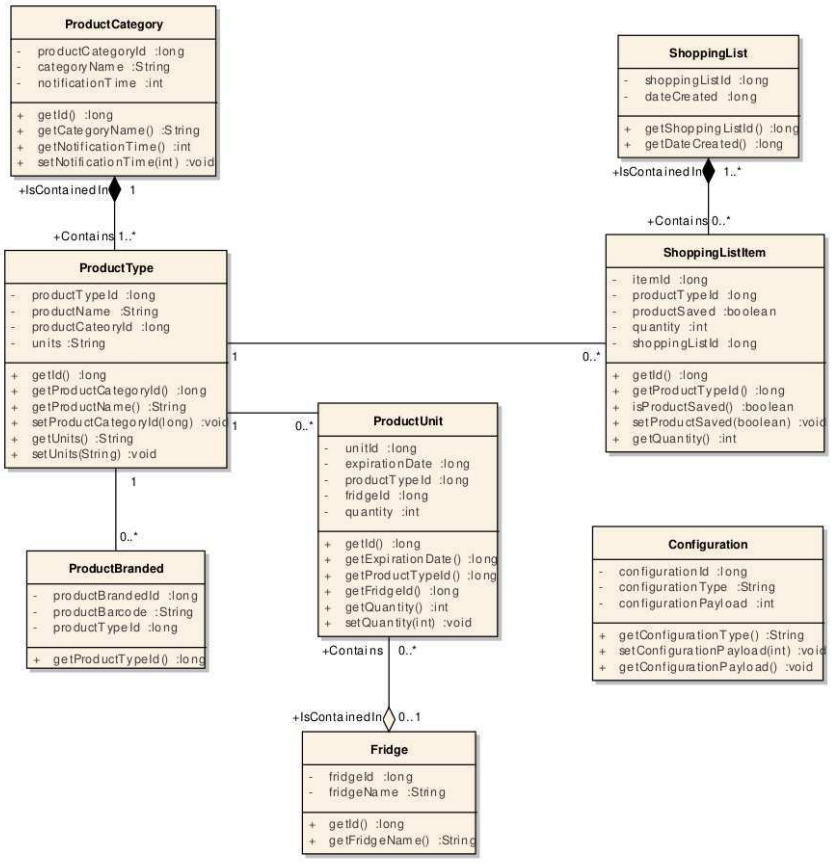
4 Návrh

Aby jsme byli schopni vyvinout robustní a smyslplnou aplikaci, je při procesu vývoje potřeba projít potřebnou fází návrhu. Naše procedura návrhu byla rozdělena na návrh databázového modelu a návrh grafického rozhraní, neboli grafický mockup. V této kapitole hodlám dopodrobna popsat tyto části a představit software, který byl použit pro ulehčení návrhu.

4.1 Databázový model

V naší aplikaci chceme permanentně přechovávat několik entit a těmi jsou produkty uložené do lednice, nákupní seznamy a konfigurace zvolené uživatelem. Výsledný návrh nebude tak jednoduchý aby nám stačili pouze 3 tabulky a tak se podrobně podívejme na jednotlivé entity.

Naší hlavní překážkou v práci s obsahem lednice bude ukládání více různých produktů pod jeden typ. Přechozí věta bude dávat mnohem větší smysl při uvedení příkladu. Tedy



Obrázek 10 Class diagram datových entit.

například máme doma 2 různé značky mléka - Madeta a Tesco Value. Uživatel při pohledu do ledničky nechce prvotně zjistit zda má dostatek mléka Madeta, ale je pro něj důležitá informace o dostatku mléka jako takového. Z tohoto důvodu jsou zavedené dvě entity ProductType a ProductBranded. Dále jsme se také v rozboru bavili o výhodnosti implementace kategorií produktů, tedy vytvoříme pro ní entitu ProductCategory a na závěr je potřeba si uvědomit, že od různých typů produktů je možné vlastnit více jednotek, které můžou mít i různé data expirace, což nás vede k zavedení ProductUnit. Také by se hodilo mít možnost oddělovat místo uložení položek, pro lehký přehled dostupných položek doma a např na chatě. Popřípadě si uživatel může jako úložiště vytvořit místa typu mrazák, spižírna a podobně. Pro tyto účely je vytvořena entita Fridge, která vlastní objekty typu ProductUnit. Tím by jsme měli mít pokrytou veškerou funkčnost ukládání položek.

Dále potřebujeme ukládat nákupní seznamy, u nichž je zajímavá pouze informace datum vytvoření pro naše interní řazení a pro logické zobrazení uživateli. Tady by se dal také implementovat název seznamu, ale osobně mi přijde takováto informace zbytečná a tudíž ji vynecháme. Pro zobrazení nákupních seznamů tedy využijeme entitu ShoppingList a pro uložení jednotlivých položek pak pro každou položku vytvoříme entitu ShoppingListItem která bude pomocí cizího klíče odkazovat na list do kterého je uložená. Shopping list item je taky spojená s prvkem typu ProductType, aby bylo možné položky rovnou z nákupního seznamu ukládat.

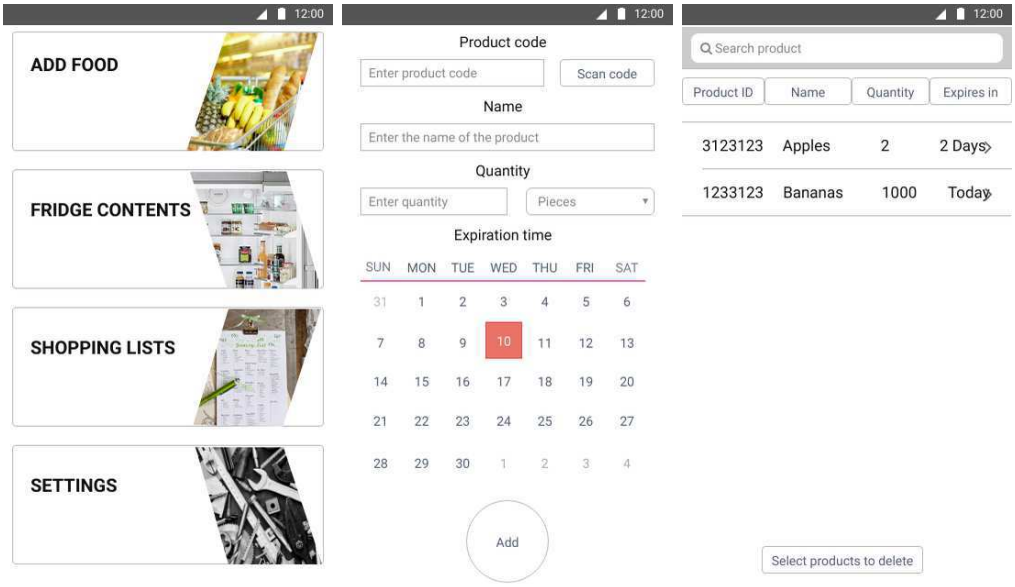
Poslední a nejjednodušší entitou jsou konfigurace, na které nám bude stačit jediná tabulka. Z pohledu databází tento přístup není správný, ale v naší situaci by nedávalo smysl vytvářet N různých entit, které by vždy měli jen jeden řádek. Aby jsme rozlišili co daný řádek konfiguruje, přidáme sloupec ConfigurationType a samotná uložená data budou v řádku ConfigurationPayload. Pro přístup do této tabulky budou ve výsledné aplikaci vytvořené soubory s konstantami, popřípadě použijeme enumy, které budou vyčítat všechny možné způsoby konfigurace, aby se nám nestalo že soubory ukládáme a vyhledáváme pod jiným textovým řetězcem. Výsledný databázový model vidíme na obrázku 10.

4.2 Uživatelské rozhraní

Výsledný vzhled pro veškeré aktivity je na obrázcích 11 a 12. Před samotným vývojem jsem se pokusil budoucí podobu aplikace nastínit pomoci webové aplikace MockingBot[1]. Už při tvorbě UI mocku bylo jasné že výsledné rozhraní bude vypadat odlišně, protože podobné neplacené aplikace na tvorbu návrhů neobsahují ani kompletně celou knihovnu Android SDK ani nedávají uživateli takové možnosti modifikace. Ale hlavním záměrem bylo nalézt potenciální chyby a obtíže v návrhu.

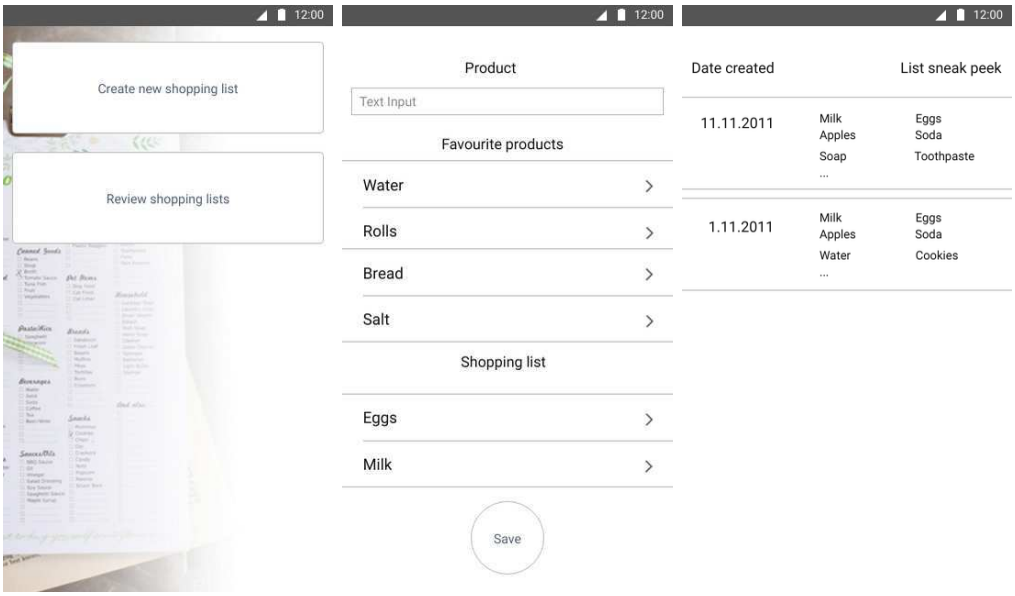
Hlavním cílem při vývoji grafického rozhraní bylo rozumně rozvrhnout funkce do jednotlivých aktivit a pokud možno nevytvářet scrollovatelné aktivity, jen v případě že zobrazujeme nějaký list informací, aby měl uživatel veškerou funkčnost na jedné obrazovce. Dále bylo cílem mít zvýrazněné klíčové funkčnosti jednotlivých aktivit. To znamená použití dostatečně velikých tlačítek jako v 11a, zvýraznění důležitého textu a rozdělení obrazovky do logických nebo chronologických sekcí jako v 11b. V rozcestníku jsou pak použity tématické obrázky pro oživení celé aplikace, ale aby grafiky nebylo zbytečně moc a rozhraní nepůsobilo zaplněně a zmateně, byly použity jen výřezky z

[1] 2017. URL: `http://www.mockingbot.com` (cit. 27. 12. 2017).



a) Rozcestník aplikace **b)** Ukládání položky **c)** Zobrazení obsahu lednice

Obrázek 11 Stěžejní aktivity FoodTrackeru



a) Rozcestnik pro nakupni seznamy **b)** Vytváření nového nákupního seznamu **c)** Historie nákupních seznamů

Obrázek 12 Nastínění práce s nákupními seznamy

původních obrázků pro minimalistický pocit. V obrázku 11c je pak naším největším nepřítelem omezené množství prostoru k zobrazení dat, naštěstí pokud by byl problém s nepřehledností, nebo pokud by se data nevešly do jednoho řádku, je možné se zbavit sloupce Product ID, který uživateli není bezprostředně důležitý.

Na obrázku 12 vidíme veškeré aktivity pokrývající funkčnost nákupních seznamů, ve výsledné aplikaci by nejspíše bylo vhodné zbavit se rozcestníku, který má stejně jen 2 tlačítka a nahradit ho například TabLayoutem. Při vytváření nákupního listu bude největším problémem zobrazit současný seznam, protože už takhle máme další listy které zabírají velikou část prostoru a uživatel by se potom mohl cítit ztraceně. U zobrazování listů v historii je opět volen minimalistický způsob zobrazení a místo celého listu je zobrazeno pouze datum vytvoření a několik prvních položek. Při kliknutí na list se zobrazí dialog se všemi položkami seznamu.

5 Vývoj

5.1 Použité technologie a software

5.1.1 Jazyky

Jedním z hlavních důvodů popularity vytváření aplikací na Android je samotný jazyk používaný na pozadí. Jedná se o Javu, mezi komunitou programátorů jde o jeden z nejpopulárnějších jazyků, který je často používán jako vstupní bod do programování. Java je objektové orientovaný, vysoce úrovňový jazyk, vyvíjený firmou Sun Microsystems od roku 1995.
Z důvodu kompatibility Androidu se zkompilovaným Javovským kódem vyvstává v poslední době další konkurent a tím je Kotlin. Kotlin je poměrné nový jazyk, poprvé představen roku 2011 [14], který získává za svojí funkčnost čím dál tím větší podporu od komunity a v dnešní době začíná být lehčí nalézt ukázky kódu Android aplikací pro Kotlin než pro Javu. Dále také vznikají pokusy jako portování nativního kódu, ku příkladu React Native, který nabízí další známý jazyk JavaScript jako variantu, nebo pomocí Android NDK je možné psát částí aplikací dokonce i v C/C++ kódu. V naší práci se omezíme na práci s Javou.

Dále bude pro práci s databází potřeba znalost SQL, nebo-li Structured Query Language. SQL je dotazovací jazyk používaný v programování pro práci s relačními databázemi. V dnešní době je už znalost SQL samozřejmostí a dost často se tato znalost nevztahuje jen na vývojáře, ale i na testery a další podpůrný personel v oblasti vývoje. Systém pro práci s databází v Android se nazývá SQLite, nicméně jeho hluboká znalost nám není potřeba a vystačíme si se základy SQL.

V poslední řadě je potřeba práce s jazykem XML - Extensive Markup Language. XML je značkovací jazyk pro kódování jednoduchých dokumentů. V Androidu je pak tento jazyk použit například pro popis rozvržení grafických elementů na obrazovce nebo na popis aplikace jako celku. Více v kapitole 5.1.4.

5.1.2 Vývojové prostředí

Jako každé jiné odvětví vývoje i Android aplikace se dají vyvíjet v jednoduchých textových editorech, najdeme zde podporu pro většinu populárních textových editorů jako je Sublime Text a například i pro vim. U větších projektů se preferuje užívání IDE, neboli integrovaného vývojového prostředí, které nám ulehčuje práci propojením s ostatními elementy vývoje Androidu, jako je Gradle, emulátory, zvýrazňování syntaxe a podobně. Vesměs každé větší IDE v dnešní době obsahuje stáhnutelný plugin na práci s Androidem, nicméně nejpoužívanějšími prostředími jsou Netbeans, Eclipse a hlavně Android Studio. My se v našem projektu obrátíme na Android Studio, které je založené na populárním vývojovém prostředí pro Javu - InteliJ Idea, ke kterému máme přístup díky studentské licenci.

5.1.3 Testovací zařízení

Pro hladký vývoj mobilní aplikace je potřeba i testovací zařízení. Zde můžeme využít buď fyzické zařízení nebo emulátor. Výhodou emulátoru je hlavně možnost otestovat aplikaci rovnou na větším množství cílových zařízení ať už v závislosti na úrovni API nebo v závislosti na velikosti displeje. Hlavními nevýhodami jsou ale známá nestabilita a také je potřeba výkonnější vývojový PC. Také v závislosti na operačním systému a počítači na kterém vyvíjíme nemusí být vůbec možné emulator rozběhat. Pro jednodušší instalaci a i lepší představu jak výsledná aplikace vypadá a možnosti vyzkoušení uživatelského zážitku z první ruky, jsem se rozhodl aplikaci testovat na fyzickém zařízení a to přesněji na Samsung S6+ Galaxy Edge. V ideálním případě by jsme chtěli testovacích zařízení víc, například pro různé rozlišení obrazovek nebo pro různé úrovně API. V našem případě by nám mělo postačit uživatelské testování provedené na konci vývoje, které potvrdí zda je naše grafické rozhraní správně navrhnuto aby cílilo všechna možná zařízení v našem vybraném rozmezí API úrovní.

5.1.4 Projektové soubory specifické pro Android

V této podkapitole probereme typické rozvržení Android projektu, tedy jaké nástroje se běžně používají a k čemu tyto nástroje jsou. Prvotní nastavení projektů může vypadat na první pohled jen jako nepříjemná aktivita, která nás zdržuje od samotného vývoje, ale špatně provedené nastavení může vést k chybám při kompilování kódu, které může trvat odbavit i několik hodin.

Android Manifest

Jedním ze základních stavebních kamenů aplikace je soubor AndroidManifest.xml. V něm jsou obsaženy veškeré informace potřebné pro operační systém telefonu pro úspěšný běh aplikace. Mezi tyto informace patří

- **Aktivity** - Seznam aktivit které se budou v programu vyskytovat a určení jedné hlavní aktivity, která má být zobrazena po spuštění aplikace.
- **Services** - Seznam procesů které aplikace vytváří a které budou běžet na pozadí, viz 2.4.
- **Oprávnění** - Některé Android aplikace vyžadují přístup k částem systému, které musí být monitorovány a které uživatel musí povolit. Jedná se například o přístup ke kontaktům nebo datům z GPS. Všechna tato vyžadovaná povolení musí být deklarována v tomto souboru.
- **Grafické specifikace** - Výsledný styl aplikace, použité logo.
- **Java package specifikace** - Pojmenování celého balíčku a cesta k němu.

Buildovací nástroj

Build tool je nástroj který se stará o výsledné vytvoření a správu vývoje celého programu. Na jednom místě se tak dají určit informace jako jsou závislosti projektu, které nám build tool je schopen i sám stáhnout po určení preferovaného repozitáře, rozmezí poskytovaných úrovní API Androidu, identifikaci aplikace a zbývající informace které jsou nezbytné pro správné vytvoření spustitelného balíčku. U Androidu převažuje podpora Gradle, který je součástí základní instalace Android Studia, nicméně je možné pomocí různých pluginů používat i jiné známé varianty, jako je například Maven. Čistě z pohledu jednodušší implementace jsme se rozhodli používat Gradle.

Layout XML Files

V podkapitole 5.1.1 jsme zmiňovali jako jednu z podmínek znalost XML a to kvůli layoutům aplikace. Layouty se skládají ze základních stavebních prvků poskytnutých z Android SDK, a pomocí nich může poměrně rychle vývojář popsat rozvržení elementů na obrazovce. Tyto layouty se používají u těch největších stavebních bloků, tedy aktivit, přes fragmenty až po různé dialogy a pickery. Tedy každé zobrazení musí mít náležitý popis pomocí .xml souboru.
Základním stavebním blokem těchto layoutu jsou tzv. view groupy které nějakým způsobem shromažďují svoje children elementy a dále s nimi pracují. Nejvíce se používají RelativeLayout a LinearLayout, kde v naší aplikaci převážně budeme používat RelativeLayout. Jeho hlavní výhodou je pozicování na obrazovce v závislosti na ostatních elementech a tím můžeme dosáhnout jednodušeji responzivního vzhledu.

5.1.5 Důležité návrhové vzory

Jen takovým doplňkem do této kapitoly je vysvětlení použitých návrhových vzorů. Návrhových vzorů je použito víc a nehodlám tu vysvětlovat vzory jako je Listener/Observer, ale rád bych objasnil pojmy DI a DAO.

Dependency Injection

Velmi zjednodušeně, místo přímého vytváření závislosti v třídě, jsou závislosti třídě poskytnuty. Poskytnuty můžou být i pomocí konstruktorů, nebo setterů, ale v našem případě budeme chtít využít specializovaných frameworků, aby tuto práci provedly za nás. Injikování pomocí setterů navíc zbytečně rozšiřuje rozhraní třídy a při velkém množství závislostí může předávání konstruktorem vytvořit dlouhé příkazy.
Přímo v Android se nabízí frameworky Roboguice a Dagger. Poslední dobou se na trh dostává i známá varianta z čisté Javy a tou je Spring. Z důvodu komplikované instalace se ale Spring v Androidu používá jen pro specifické případy a jelikož Roboguice není nadále vyvíjen a je deprekován, jsme se rozhodli pro využití Daggeru. V kombinaci s Daggerem se dá ještě použít další DI framework s názvem Butterknife. Ten je úzce specializovaný na práci s grafickým rozhraním a umožňuje nám přehledněji pracovat s elementy rozhraní.

Database Access Object

DAO [15] je další zajímavý vzor který odděluje práci s datovou vrstvou od práce s logickou vrstvou. Pro nás je obzvláště kritický, protože převážná část aplikace je založená na práci s databází. Pro oddělení máme následující třídy

- DAO Rozhraní - Definuje potřebné funkce.
- DAO Konkrétní implementace - Implementuje třídy předepsané v rozhraní.
- Entita - POJO objekt použitý k uložení dat.

Díky tomuto rozdělení dostáváme do naší implementace další vrstvu a v případě nutnosti změny v nejnižších vrstvách je potřeba provést opravy dál jen v naší DAO vrstvě. Logická vrstva díky tomu zůstává nezasažená.

Pro práci s DAO budeme v našem projektu používat framework ORMLite. Ormlite je knihovna na objektově relační mapování - ORM, která nám nabízí následující funkce

- Definování entit datové vrstvy.
- Rozhraní pro práci s DAO objekty.

- Zpracování transakcí mezi logickou vrstvou a datovou.
- Popisuje vztahy mezi entitami.
- Vytváří SQL dotazy do databáze.

ASF

Přímo pro práci s ORMlite pak používáme další knihovnu ASF [16], která nám nabízí rozhraní pro generické vytvoření vrstev pro práci s DAO objekty. Díky této knihovně máme k dispozici defaultní entitu, od které můžeme dědit naše vlastní datové entity, které díky dědění získají přístup ke generování primárního klíče a funkce s ním spojené. Dále získáváme další vrstvu mezi logickou vrstvou a datovou, která se stará o práci s výjimkami a o správné provedení práce s DAO vrstvou. Knihovna ASF je dodávaná od vedoucího bakalářské práce pana Balíka.

5.2 Testovací projekty

Před finální fází vývoje aplikace jsme se rozhodli že by bylo nejlepší si některé elementární části FoodTrackeru naimplementovat jako samostatné a jednoduché aplikace. Na těchto aplikacích by jsem se jako začátečník v tomto oboru naučil postupy a ověřil zda je náš přístup správný. Také by nám to výrazně ulehčilo skládání výsledné aplikace a testování jednotlivých částí. Mezi testovací projekty patří :

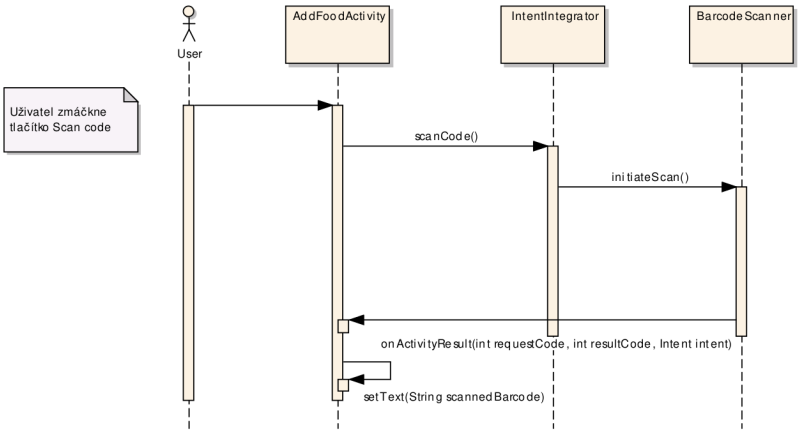
- Čítač čárových kódů
- Posílání notifikací
- Práce s databází a později rozdělení databáze na několik vrstev - DAO.

5.2.1 Čtení čárkových kódů

Jelikož nebylo cílem této aplikace vyvíjet vlastní čítač, rozhodli jsme se použít open-source produkt BarcodeScanner[1] od developerského týmu ZXing, který je určitým standardem načítání čárkových kódů v oblasti Android aplikací. Jediným požadavkem pro úspěšné fungování je si stáhnout tuto aplikaci z trhu Google Play a FoodTracker bude poté pomocí API předávat dotazy na načítání přímo této aplikaci. Pro zjednodušení je uživatel při prvním používání dotázán zda chce opravdu tuto funkci využívat a zda je ochoten stáhnout další aplikaci a poté je přesměrován na stránku Barcode Scanneru na Google trhu.
Naštěstí pro nás, jejich API je již časem ověřené a vše co je potřeba k rozběhání této funkce jsou 2 soubory. Jako grafické rozhraní nám postačí pouze tlačítko na spuštění scanu a TextView na zobrazení čísla čárkového kódu. Klíčové jsou pouze 2 funkce které volají API barcode scanneru. Komunikaci můžeme popsat sekvenčním diagramem, který je na obrázku 13

IntentIntegrator je přímo třída poskytnutá ve frameworku, která pro práci potřebuje jen současný kontext aplikace. Součástí zavolání funkce initiateScan() je i ověření že uživatel má k dispozici aplikaci Barcode Scanneru a v případě že tuto aplikaci zatím nevlastní je v případě zájmu přesměrován na potřebnou stránku obchodu Google Play.

[1] *ZXing ("Zebra Crossing") barcode scanning library for Java, Android.* URL: `https://github.com/zxing/zxing` (cit. 10. 05. 2018).



Obrázek 13 Sekvenční diagram scanování kódu.

5.2.2 Posílání notifikací

Žádaným výsledkem této části byla opět jednoduchá aplikace, s jediným tlačítkem, která po zmáčknutí vytvoří nějakou notifikaci a okamžitě ji odešle. Ve finálním produktu bude samozřejmě posílání notifikací o něco složitější. Na pozadí budeme potřebovat service, která bude hlídat zda některé produkty nepřekročili hranici expiračního data kterou si uživatel nastavil. Pro naše účely nám ale postačí zjednodušená verze. Dokonce je kód tak jednoduchý že se nám podařilo ho vměstnat do jediné funkce

```
public void sendNotification(View view) {
    NotificationCompat.Builder notBuilder =
     new NotificationCompat.Builder(this);
    notBuilder.setSmallIcon(R.mipmap.ic_launcher);
    notBuilder.setLargeIcon(BitmapFactory
        .decodeResource(this.getResources(),
        R.mipmap.ic_launcher));
    notBuilder.setContentTitle("Testing notification");
    notBuilder.setContentText("Hello there!");
    Intent resultIntent = new Intent(this, MainActivity.class);

    TaskStackBuilder stackBuilder =
     TaskStackBuilder.create(this);
    stackBuilder.addParentStack(MainActivity.class);
    stackBuilder.addNextIntent(resultIntent);
    PendingIntent resultPendingIntent =
    stackBuilder.getPendingIntent(
        0,
        PendingIntent.FLAG_UPDATE_CURRENT
        );
    notBuilder.setContentIntent(resultPendingIntent);
    NotificationManager mNotificationManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);

    mNotificationManager.notify("Notification Test",
        1, notBuilder.build());
```

```
}
```

K vytváření notifikací je v Androidu připravený builder, který získáme zavoláním jeho konstruktoru a dodáním kontextu. V příkladu vidíme i možnost přidání ikony do náhledu notifikace a poté je nastaven titulek a samotný text notifikace. Poté vytvoříme Intent vedoucí zpět k naší hlavní aktivitě, který později přidáme na StackBuilder. StackBuilder je uměle vytvořený zásobník, který předáme vytvořené notifikaci, tak aby když klikneme na notifikaci se otevřela hlavní aktivita aplikace a po zmáčknutí tlačítka zpět jsme byli převedeni na home screen. Na závěr získáme systemovou service na vytváření notifikací a vytvoříme notifikaci pomocí dříve vytvořeného builderu a následně odešleme pomocí manageru.

5.2.3 Práce s databází

Součástí toho projektu byly 2 aplikace, kde v té první bylo cílem pouze vytvořit datovou vrstvu a pomocí SQL dotazů přistupovat k databázi. Tento přístup je ale nepraktický, obsahuje větší množství kódu a SQL queries jsou uloženy jako stringy přímo ve třídě ve které probíhá volání do databáze, což působí nepřehledně a jde proti objektovému přístupu raženému v Javě a taktéž v Androidu. Raději bych tedy rovnou představil druhý projekt který databázi implementuje jako Data access object. Nejdříve si nadefinujeme naší databázovou entitu, v tomto případě se jedná o produkt který má svůj unikátní klíč Id a dále název či popis produktu nazvaný Description a množství produktů Qty. Takto navržený produkt nesouhlasí s naším databázovým návrhem finální aplikace, ale pro učící a testovací účely postačí. Samotná definice třídy vypadá následovně

```
@Entity(name = "Product")
public class Product extends AbstractPersistable {
    private static final long serialVersionUID = 1L;

    @Column(name = "Description", nullable = false)
    private String description;

    @Column(name = "Qty")
    private int qty;

    public Product(ProductBuilder builder) {
        setId( builder.getId());
        this.description = builder.getDescription();
        this.qty = builder.getQty();
    }

    public Product() {
        this(0, "", 0);
    }
    ...
}
```

Tři tečky na konci souboru skrývají konstruktory, gettery a settery, které pro nás v tuto chvíli nejsou zajímavé. Naše entita rozšiřuje třídu AbstractPersistable z frameworku ASF, její funkce je poskytnout třídu která má vlastní ID a pomocné gettery

na rozšíření entit jako je například náš Product. Místo ORMlite anotací jsou použity JPA anotace pro více obecný přístup. Vidíme že ve třídě jsou definovány potřebné sloupce description a qty a k nim potřebné gettery a konstruktory. Zajímavá je nutnost použití defaultního konstruktoru pro interní použití ORMlite frameworku. Sloupec Id je děděn z AbstractPersistable jak je vysvětleno výše. Persistentní vrstva je tímto hotová, nyní je potřeba napsat vrstvu která se bude starat o práci s těmito daty, zvanou repozitář. Tou třídou je pro nás ProductDao, která rozšiřuje další třídu AbstractDAO opět z frameworku ASF.

```
public class ProductDao extends AbstractDAO<Product> {

    @Inject
    public ProductDao(DbHelper dbHelper) {
        super(dbHelper.getConnectionSource());
    }
}
```

Třída ProductDao díky námi zvolenému přístupu pak vypadá takto jednoduše, veškerá logika je skrytá ve třídě AbstractDAO a je možné ji použít genericky i pro ostatní entity. Abstract DAO poskytuje CRUD operace a schovává nepřehledné části kódu ve kterých je například odchytávání chyb vzniklých nenalezením potřebného elementu v databázi. K práci této vrstvy je potřeba OrmliteSqlite helper, který dědíme do třídy DbHelper. Ve třídě DbHelper jsou definovány informace jako je název a verze databáze a také ConnectionSource, což je třída která vytváří spojení mezi repozitářem a přístupovou vrstvou. Pro samotné CRUD operace je pak použit DaoManager dodáváný z balíčku OrmLite.
Součástí této testovací aplikace je také poměrně rozšířené uživatelské rozhraní, ale není potřeba ho tu představovat jelikož podobný kód bude použit i ve finální aplikaci.

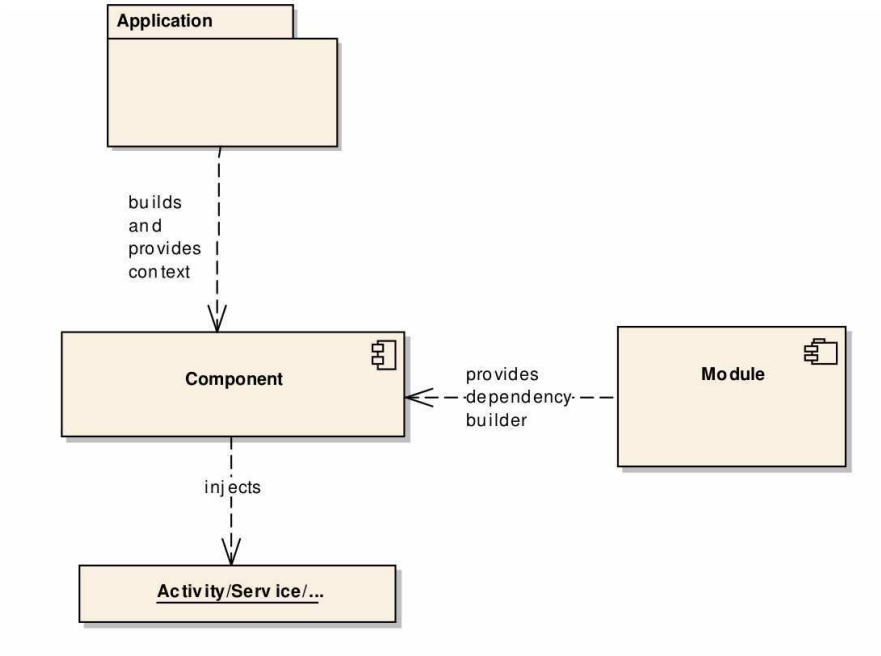
5.3 Implementace FoodTrackeru

5.3.1 Základ aplikace

Když máme připraveny hlavní elementy z testovacích projektů, je na čase začít dávat dohromady výslednou aplikaci. Pro přehlednější implementaci je v projektu použit dagger, jehož funkce byla vysvětlena v kapitole 5.1.5, a pro jeho využívání je nutné při prvním spuštění vytvořit potřebné třídy které se budou starat o injikování závislostí. Jako vstupní bod vytvoříme následující třídu rozšiřující objekt Application.
Aby tato třída byla aplikací považována za vstupní bod, musíme ji zaregistrovat do AndroidManifestu následujícím způsobem

```
android:name=".FoodTrackerApplication"
```

V některých případech použití daggeru by tato třída nebyla nutná, ale v našem případě chceme, aby se nám samy inicializovaly třídy, které k jejich inicializaci potřebují kontext aplikace. Kontext aplikace jim dodáme pomocí třídy která samotnou aplikaci rozšiřuje, ale ještě potřebujeme zaregistrovat tzv. provider, který bude vědět kde tento kontext najít. K zaregistrování výše zmíněného providera potřebujeme Module a Component. Componenta musí být zaregistrována jako singleton, aby nedocházelo k inicializaci více než jedné instance. Výsledný diagram souvislosti je zobrazen na obrázku 14.



Obrázek 14 Komunikace mezi komponenty dagger frameworku.

Obrázek 15 Rozcestník aplikace

5.3.2 Grafické rozhraní

U grafického rozhraní navazujeme na návrh v sekci 4.2, ale i tak jsou zde provedeny nějaké změny, které většinou vznikli v návaznosti na osobní testování. Při vývoji uživatelského rozhraní jsme se omezili na Android SDK, i přes vzrůstající popularitu tvorby rozhraní za pomoci React Native a to čistě z důvodu časové náročnosti. Jako doplněk jsme použili podpůrnou knihovnu Materials Design[2] od Google, která se v tomto odvětví stává standardem. Proniká totiž nejen do trhu aplikací Android ale i do kon-

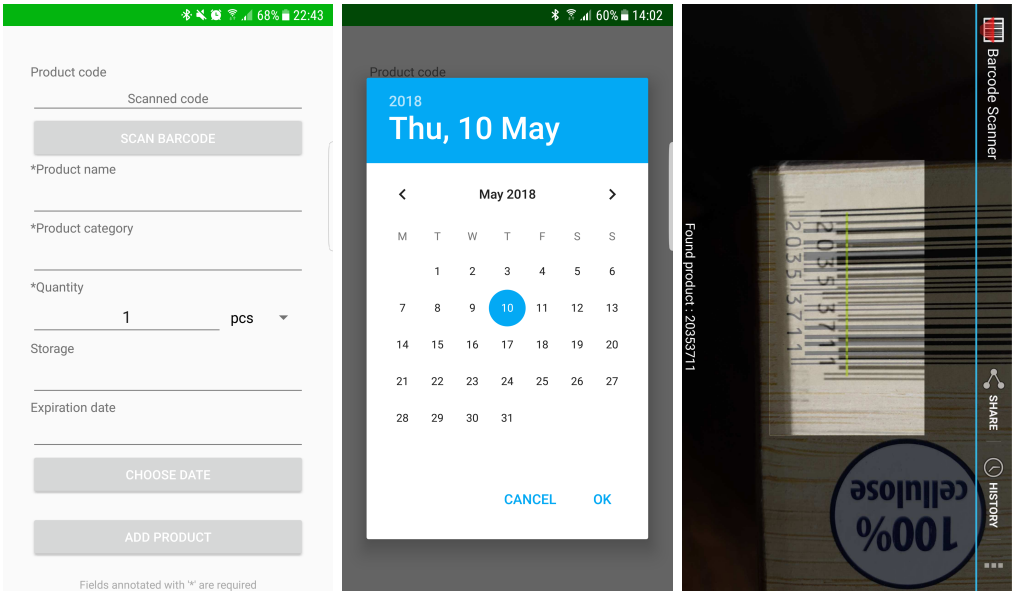
[2] *Material design.* URL: `https://material.io/` (cit. 10. 05. 2018).

kurečního iOS trhu a taktéž jsou jejich prvky používány při vývoji webů. Kromě sbírky dobře vypadajících elementů zde totiž najdeme i články vysvětlující do hloubky způsoby návrhu uživatelského rozhraní a příklady ukázané na návrzích grafických elementů z profesionálních aplikací. Grafické rozhraní je prostředí ve kterém by se dalo strávit mnohem více času a pokud bychom usilovali o finální vydání aplikace na trh stálo by za to pokusit se ho vypilovat. Z původního návrhu se nám skoro přesně zachoval rozcestník aplikace. I přesto že Google a moderní aplikace razí heslo pouze jediné aktivity a zbytek funkčnosti schovat do fragmentů, myslím si že v našem případě pořád rozcestník dává smysl a dodává uživateli smysluplné rozdělení aplikace na několik víceméně samostatných modulů. O těchto modulech budou pojednávat následující podkapitoly, a začneme hlavní aktivitou aplikace, která funguje jako rozcestník.

```
public class MainActivity extends AppCompatActivity {

    @BindView(R.id.addFoodButton)
    Button addFoodButton;

    ...

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ButterKnife.bind(this);
    }

    @OnClick(R.id.addFoodButton)
    void moveToAddingFood() {
        Intent intent = new Intent(MainActivity.this,
            AddFoodActivity.class);
        intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        startActivity(intent);
    }

    ...
}
```

Díky frameworku ButterKnife se nám podařilo dosáhnout takto přehledného kódu. Pomocí tohoto frameworku jednoduše nalezneme jednotlivé elementy uživatelského rozhraní a dokonce jim můžeme pomocí anotací přidat i posluchače, jak je ukázáno u funkce moveToAddingFood(), místo toho aby jsme museli pracně implementovat funkční rozhraní, které zbytečně roztahuje kód. Celá aktivita obsahuje pouze tlačítka pro přechody mezi potřebnými aktivitami, jejichž kód je ve stejném stylu jako v ukázce. Takto tedy nadefinujeme přechody do sekcí s ukládáním položek, zobrazování položek, práci s nákupními seznamy a nastavení. Použité obrázky jsou dostupné ke komerčnímu využívání a všechny byly staženy z platformy Pexels[3]. Nyní je čas se podrobně podívat na jednotlivé aktivity.

[3] *Pexels - Best free stock photos in one place.* URL: `https://www.pexels.com/`.



a) Ukládání položky. **b)** Výběr data expirace **c)** Načítání čárkového kódu

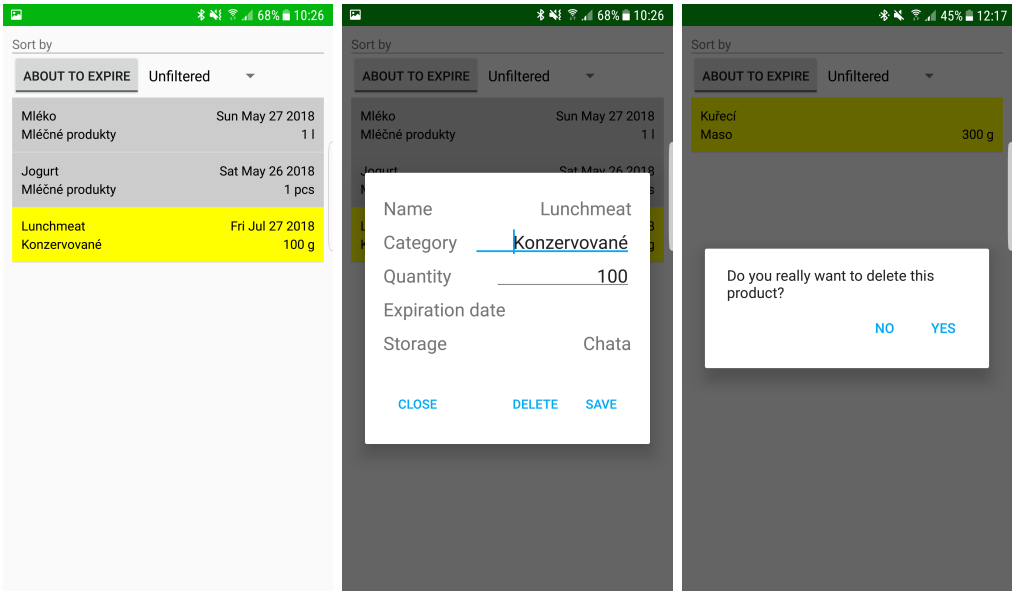
Obrázek 16 Aktivita pro přidávání položek se všemi zobrazitelnými fragmenty.

Ukládání položek

Jak vidíme na obrázku 16a, pro přidání položky je použita jediná aktivita která pro úspěšné uložení potřebuje název přidávaného produktu, kategorii do které má být zařazen a jeho množství. Tyto položky jsou anotovány hvězdičkou a popisek je udán na spodu obrazovky, který informuje uživatele o tom, že další informace nejsou nezbytně nutné. Čárový kód sice značně urychluje práci při načítání, ale není možné ho používat vždy. To samé platí pro datum expirace, u některých položek nemusí nutně být datum známo nebo nemusí vůbec existovat.
Jednotlivé políčka za sebou jdou v chronologickém pořadí, tedy pokud uživatel vyplní nějaké políčko, políčka jdoucí po něm se předvyplní pokud je databázi produkt již znám. Pro příjemné načítání expiračního data je vytvořen fragment kalendáře, výsledek je zobrazen na obrázku 16b. Dále je ke zrychlení načítání ke každému textovému vstupu připojen autocomplete, jež doplňuje rozepsaný text podle relevantních položek které nalezne v databázi. Hlavní problém při rozvržení rozhraní byl u této aktivity v tom, že podle zásad by se všechny akční elementy měly dát do spodu obrazovky, přesněji tak aby byly v dosahu palce u mobilního uživatele. Na druhou stranu při přesunutí tlačítka na skenování do spodu obrazovky by jsme porušili systém chronologického seřazení. Čistě v zájmu udržení konzistence rozhraní jsem se tedy nakonec rozhodl nechat tlačítko v horní části.

Zobrazování položek

K zobrazení velkého množství informací jsme použili ListView, k jehož správnému fungování je potřeba definovat jak vypadá jeden řádek seznamu a nakonec k celému seznamu napsat ListAdapter. Základem je rozšíření třídy BaseAdapter, které nám nutí implementovat několik getterů, které můžeme uspokojit zavedením interního použití jakékoliv datové struktury, od arrayů, přes listy až po mapy. Pro interní práci s produkty je definována třída ProductView, která shromažďuje veškeré potřebné informace



a) Zobrazení uložených položek

b) Detail zobrazen při vybrání položky

c) Upozornění při mazání produktu

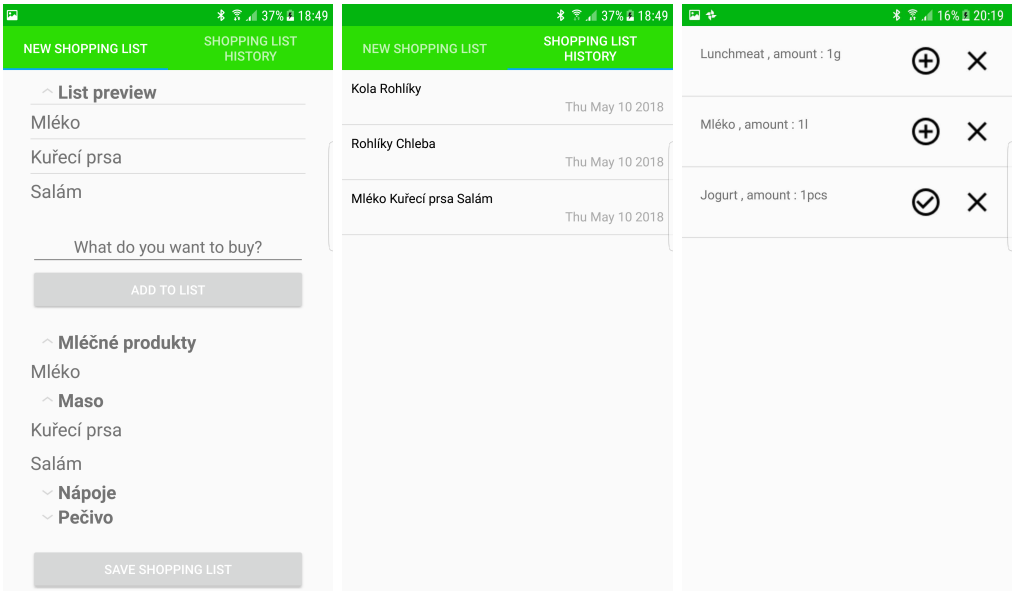
Obrázek 17 Rozhraní pro práci s obsahem lednice.

na jednom místě a nepotřebujeme všechny 4 entity které se jinak starají o ukládání dat o produktech. Poté už se jen ve přepsané funkci getView() stačí postarat o naplnění řádku tabulky předanými daty v konstruktoru.

Dále bylo u zobrazení lednice kritické, aby uživatel mohl s daty nějak interagovat, mazat staré a prošlé produkty, popřípadě měnit kvantitu uložených produktů. K tomu byl vytvořen dialog zobrazený na obrázku 17b a spolu s ním vzniknul ten problém, že při smazání položky nebyl uvědomen list uchovávající data. Pro upozornění jsem použil návrhový vzor Observer/Listener. Aktivita starající se o lednici pak rozšiřuje rozhraní Refreshable, které předepisuje funkci refreshListAdapter.Tento přístup je použit i v ostatních aktivitách s listy které potřebují být obnovovány. Při vytvoření dialogu na mazání prvků se zaregistruje aktivita jako posluchač a v případě smazání prvku je obnoveno zobrazení listu. Součástí mazacího dialogu je také AlertDialog který se od uživatele ujišťuje o tom že opravdu chce položku smazat.

Poslední důležitou funkcí je řazení seznamu. V původním mockupu je mezi způsoby řazení i řazení podle abecedy, ale pro přehlednost zůstali pouze dvě řadící funkce, které jsou pro uživatele nejdůležitější. Jedná se o řazení podle blížící se expirace a o řazení podle úložiště. Tlačítka na sobě mají listenery, které vytvoří potřebné komparátor. Tento komparátor je pak předán funkci sortWithComparator() která po seřazení listu zavolá funkci refreshListAdapter().

```
private void sortWithComparator(Comparator comparator) {
        Collections.sort(productsView, comparator);
        ProductListAdapter productListAdapter =
        new ProductListAdapter(getApplicationContext(),productsView);
        productList.setAdapter(productListAdapter);
    }
```



a) Vytváření nového nákupního seznamu.

b) Historie vytvořených seznamů.

c) Detail jednotlivých nákupních seznamů.

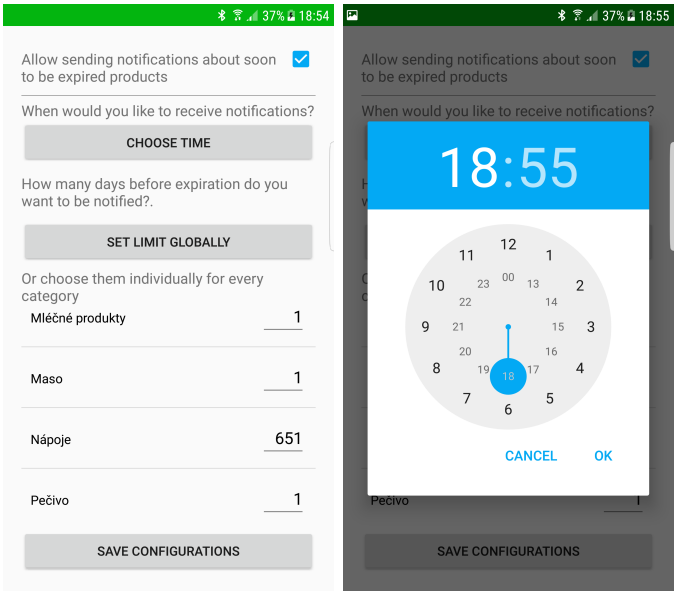
Obrázek 18 Rozhraní pro práci s nákupními seznamy.

Řazení podle úložiště je vytvořeno elementem zvaným Spinner, který obsahuje několik možností zobrazitelných při kliknutí na tento element. Vybraná možnost je pak zobrazena při zavřeném

Nákupní seznamy

U aktivity pro nákupní seznamy jsem se oproti mockupu odchýlil od dalšího rozcestníku který obsahoval jen 2 tlačítka a nahradil ho TabLayoutem z knihovny MaterialsDesign[4]. Veškerá funkčnost tak stále zůstává na jednom místě, ale působí přehledněji. Opět se zde setkáváme s problémy řešenými v minulých sekcích. Bylo potřeba vytvořit Listener který se stará o aktualizaci dat v listu a vytvořit adapter který bude zobrazovat data. Tento adaptér je o něco složitější, jelikož se jedná o ExpendableListView, který nám nabízí navíc funkci zavírání a otvírání jednotlivých kategorií listu. Ale logika implementace zůstává stejná. Při textovém zadávání je opět samozřejmostí autocomplete a jako reakce na zmáčknutí tlačítek jsou dodány toasty pro ujištění uživatele že se akce zdařila. Vytváření nového listu je tedy spíš o kontrolování chování uživatele, zda se nesnaží přidat do listu duplicitní položky, zda není název produktu prázdný a podobně.

V historii nákupních seznamů je vše stejné jako v návrhu, zobrazen je pouze datum vytvoření seznamu a je zobrazeno několik jeho prvních položek. Všechny položky pak můžou být zobrazeny kliknutím na příslušný seznam a objeví se nám aktivita jako je na obrázku 18c. Pomocí plus tlačítek vedle názvů produktů si uživatel může vybrat produkty které již nakoupil a rovnou je uložit do databáze, s tím že má již předvyplněna data dostupná z položky nákupního seznamu.



a) Přehled všech nastavení aplikace

b) Vybírání času pro příjem notifikací

Obrázek 19 Aktivity pro konfiguraci chování aplikace

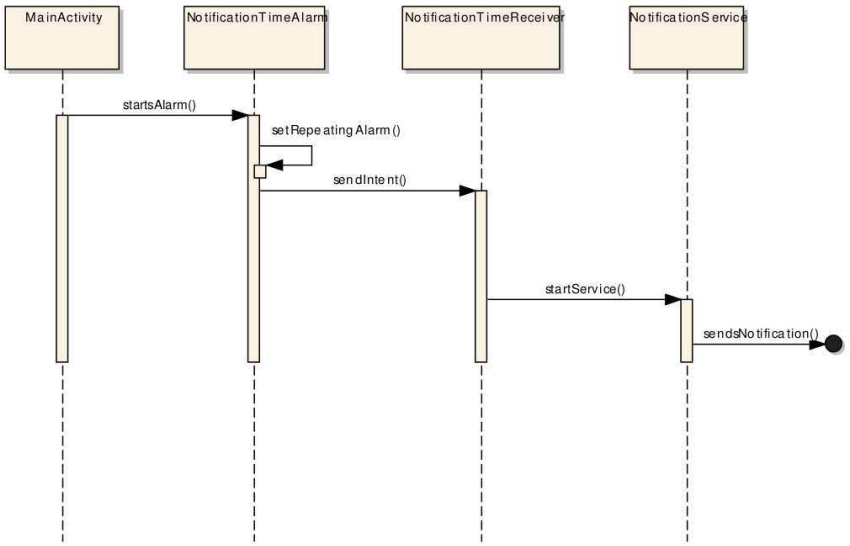
Konfigurace

Veškeré informace které potřebujeme konfigurovat od uživatele jsou časové limity, ve kterých by chtěl být notifikován o expirujícím produktu a zda by vůbec chtěl být notifikován. U některých aplikací nejsou vypínatelné notifikace a uživatelé si dost často v komentářích stěžují, proto tato volba. Časový limit je nastavován pro celé kategorie jak bylo zmíněno v rozboru funkčnosti. Uživatel má taky možnost si vybrat kdy mu tyto notifikace budou chodit, aby se nestávalo že notifikace přijdou někdy v nevhodnou chvíli. Pro výběr času je použit TimePicker jako je na obrázku 19b. Kontrolování databáze pro zastaralé prvky pak proběhne jen jednou denně ve vybranou dobu, přijde mi z hlediska návrhu zbytečně tyto expirace kontrolovat častěji. Problém vyvstává v případě vypnutí telefonu. Service totiž nebyla zařazena mezi procesy spouštěné při zapínání telefonu a je obnovena až při dalším otevření aplikace.

5.3.3 Práce s databází

Práce s databází je rozvržena do několika vrstev, těmi jsou datová, přístupová a servisní vrstva. V datové vrstvě jsou definovány používané entity stejným způsobem jako v 5.2.3. Tyto entity jsou všechny zobrazeny v diagramu 10. Přístupovou vrstvou pak považujeme vrstvu která přímo spravuje a ukládá tyto entity. O celou přístupovou vrstvu je postaráno díky frameworku ASF. Servisní vrstva používá funkce z přístupové vrstvy a dále je obaluje do funkcí které jsou bližší vývojáři. Například kde přístupová vrstva komunikuje s databází pomocí funkcí queryForEq() nebo findById(), servisní nabízí už přímé implementace a název by potom byl například findCategory() nebo findCategoryByName(). Přístup je stejný jako v kapitole 5.2.3 a jediné v čem se liší finální aplikace je velikost rozhraní servisní vrstvy a množství datových tříd.

[4] *Material design.* URL: `https://material.io/` (cit. 10. 05. 2018).



Obrázek 20 Sekvenční diagram posílání notifikací.

5.3.4 Posílání notifikací

Pro posílání notifikací jsme museli vytvořit service která poběží na pozadí. Jelikož se nejedná o složitou operaci a postačí nám k vykonávání jedno vlákno, je výhodnější třídu dědit od předka IntentService. Tato třída má přístup do databáze a v případě že je spuštěna, přefiltruje všechny produkty a zjistí které z nich jsou již pod expirační hranicí. Pokud jsou nějaké takové produkty nalezeny, je poslána notifikace, podobně jako v 5.2.2, jen s tím rozdílem že je využíváno navíc roztáhnutelných notifikací a je přidána ikona FoodTrackeru. Roztáhnutelné notifikace jsou takové, které mají 2 různé stavy, v jednom je pro jednoduchost zobrazen pouze titulek a ve druhém stavu je zobrazeno větší množství textu. Bylo nutné použít takovéto dvoustavové notifikace, protože se dá předpokládat že množství produktů které budou brzy expirovat může dosáhnout vysokého počtu.
Ke spuštění service je používána třída rozšiřující BroadcastReceiver [20].

```
public class NotificationTimeReceiver extends BroadcastReceiver{

    @Override
    public void onReceive(Context context, Intent intent) {
        Intent i = new Intent(context,NotificationService.class);
        context.startService(i);
    }
}
```

Tato třída již z názvu naslouchá případnému dotazu o spuštění vlákna a v případě nalezení takového dotazu mu vyhovuje. K naplánování času notifikace je napsána třída NotificationTimeAlarm.

Tato třída při zavolání její funkce použije uložená nastavení a podle nich naplánuje čas poslání zprávy kterou může odchytit dříve zmíněný NotificationTimeReceiver. Pro jednodušší představení můžeme zobrazit celou komunikaci na sekvenčním diagramu 20

6 Testování

Jak bylo řečeno v úvodní kapitole, součástí této kapitoly je uživatelské testování. Bude ukázána statistika zúčastněných uživatelů, jejich hodnocení a připomínky získané na základě dat z dotazníků. Pokusím se na vzniklé připomínky i nějak reagovat, ať už rovnou opravou nalezeného problému, či alespoň navrhovaným řešením do budoucích verzí.

6.1 Uživatelské testování

Před testováním všichni uživatelé vyplnili pretestovací dotazník, pokládané otázky obsahovaly :

- Jak dlouho používáte smartphone?
- Jak dlouho používáte Android?
- S jakou intenzitou denně využíváte váš smartphone?
- Věk, pohlaví.
- Chtěl/a by jste aplikace typu FT využívat pravidelně?
- Jak staré je vaše zařízení (rok uvedení na trh)?
- Jak dlouho používáte vaše zařízení (rok koupě)?

Uživatelského testování se zúčastnilo 12 různých uživatelů, z nichž jsou 2 ženy a zbytek muži. Bohužel se nám nepovedlo zacílit i uživatele starších zařízení, nejstarší zařízení z této skupiny bylo vyrobeno roku 2013. I rozlišení obrazovek bylo v podobném rozmezí s naším testovacím zařízením, proto jediný způsob kterým ověříme funkčnost těchto zařízení bylo pomocí emulátoru v Android Studiu. Naše aplikace prošla úspěšně na obou extrémech velikosti rozlišení a o případné porušení cílového API se taktéž stará naše IDE, které okamžitě upozorňovalo na potenciální chyby při použití příliš nových technologií.
V testování byly zasaženy byly skoro všechny věkové kategorie, nejmladšímu uživateli bylo 12 let a nejstaršímu 56, s mediánem v 23 letech. Většina testerů používá smartphony pravidelně a poměrně intenzivně, takže by ani nemělo vadit nedostatek technického pozadí u většiny dotazovaných, naopak můžeme vyzkoušet jak se bude na aplikaci tvářit běžný uživatel.
Ve skupině byla polovina dotazovaných proti používání aplikace typu FoodTracker, což by šlo proti výsledkům z naší analýzy trhu. Ovšem tento problém téměř bez výjimky vznikal u uživatelů, kteří se dosud nemusí pravidelně starat o běh domácnosti a tak starší uživatelé, kteří se již starají o rodiny byli nápadem osloveni.

Testovacími úkoly bylo pouhé využívání aplikace při naskytnutí příležitosti. Tedy pokud možno pokaždé před nákupem vytvořit nákupní seznam a poté všechny položky načíst do databáze. Dotazovaní byly i vyzvání k využívání notifikací a pokud možno i spravování databáze. Po dokončení testování byly sesbírány i post-testovací dotazníky, které obsahovali následující otázky

- Jak dlouho jste aplikaci používal/a? (čistý čas strávený uvnitř aplikace)

- Ohodnoťte následujicí aktivity podle uživatelského zážitku (přidávání položek, prohlížení lednice, vytváření nákupních seznamů, procházení nákupních seznamů, nastavení
- Byly nalezeny nějaké nesouvislosti v aplikaci (bugy)?
- Jaké funkce by jste si představovali v dalších verzích? - Podle aktivit
- Chtěl by jste po vyzkoušení využívat aplikaci FT?

Sesbírané informace by se dali rozdělit do několika kategorií. Těmito kategoriemi budou uživatelské rozhraní, výkonové problémy, bugy a vyžadování dalších funkcí. Naštěstí k výkonovým problémům u žádného z uživatelé nedošlo a ani by nastávat neměly, protože náš databázový model neobsahuje nijak závratné množství dat.

6.1.1 Bugy

Při testování uživatelé naštěstí narazili pouze na 2 bugy. Jedním bylo padání aplikace při otevření kategorie, která již neobsahuje žádné typy produktů. Oprava tohoto problému naštěstí nebyla složitá, jednalo se o nullpointer exception v jednom z adaptérů. Druhým problémem bylo načítání čárových kódů, které obsahovaly i pomlčky. Problém vyplýval z mého databázového návrhu, kde jsem předpokládal že všechny čárové kódy se dají charakterizovat datovým typem long, stačilo změnit funkce a model na práci se stringy.

6.1.2 UI/UX Testování

Reakce na celkový pocit z ovládání aplikace byly kladné. Toto hodnocení by se dalo připsat i subjektivitě uživatelů, v profesionálním prostředí by jsme se určitě dočkali otevřenější kritiky. I tak se mi ale povedlo z dotazovaných získat několik cenných připomínek, které bych tu rád rozebral a v případě dalšího vývoje na ně nějak reflektoval.

Práce s obsahem lednice

Nejčastěji se objevovala připomínka na způsob orientace v obsahu lednice. Uživatelé chtěli mít možnost vyhledávání podle určitého řetězce a více možností řazení, hlavně vyfiltrovat pouze expirované produkty. Řešením by tedy bylo přidat search bar a napojit na něj adaptér s názvy jak názvů produktů, tak s názvy kategorií a značek. Tato funkce byla původně v grafickém mocku načrtnuta, ale v průběhu implementace jsem se mylně domníval že by ji většina uživatelů používat nechtěla.
K adresování druhého problému by stačilo udělat tlačítko na expiraci produktů dvoustavové a podle jeho stavu filtrovat pouze produktu které brzy budou expirovat a v druhém stavu zobrazit nejdříve expirované produkty.
S tím souvisí i další připomínka a tou je nemožnost mazání většího množství položek z databáze. Mazání po jedné položce bylo příliš zdlouhavé a taktéž byla i změna kvantity. Jako reakci na tento problém mě napadá použít nějaký z pokročilejších způsobů zobrazení listů, například karty z knihovny Materials Design, které umožňují lepší obratnost při umisťování elementů na které se dá klikat do položek listu. Do této karty by jsme pak například přidali checkbox pro hromadný výběr a nějaký number picker který by umožňoval okamžitě měnit kvantitu. U klasického ListView jsem dlouho řešil problém, kdy Listener na editText nezareagoval na kliknutí kvůli charakteristikám jeho rodičovského objektu, tyto problémy by se snad použitím karet vyřešili.

Práce s nákupními seznamy

U nákupních seznamů jsem se hlavně setkával s připomínkami ohledně špatného zobrazení současného listu. Při zavedení většího množství kategorií byla aktivita příliš přeplněná a špatně se v ní orientovalo. Takový problém by se nejspíš dal vyřešit rozdělením problému na více obrazovek. Například by jsme měli aktivitu ve které by se zobrazoval aktuální seznam pomocí ListView a dospod by jsme dali tlačítko na přidání položek do listu. Teprve po zmáčknutí tlačítka by se otevřel fragment na kterém by jsme mohli rychle přidávat položky podle jejich kategorií nebo podle popularity, dále by tu bylo textové zadávání a v ideálním případě i hlasové přidávání.
U zobrazování seznamů uživatelé chtěli nějak zvýraznit poslední vytvořený seznam, tedy ten aktuální podle kterého jdou zrovna nakupovat. Jedná se o maličkost, ale takovéto maličkosti dost často přidávají nejvíce k uživatelskému zážitku. Tento problém by se dal vyřešit zase nějakými kartami, a první seznam by byl zvýrazněn tak, že by byl roztažený a ukazoval všechny položky.

Nastavení

V nastavení si uživatelé stěžovali na velké množství informací v textu a aktivita opět působila přeplněně. K tomuto problému jsem došel i při vytváření rozhraní a řešením k němu by bylo přidání nějakého plovoucího tlačítka, například s ikonou otazníku, značící pomoc. Po zmáčknutí by se pak otevřel nějaký fragment zobrazující veškeré potřebné informace a vysvětlivky.

Závěr z testování

Naštěstí se nám z uživatelů povedlo i přes kladné reakce dostat nějaké užitečné informace, které můžeme použít k budoucímu zdokonalování aplikace. Nabízela by se i možnost kompletního refaktoringu vzhledu aplikace, aby aplikace vypadala více moderně s menu na straně obrazovky a místo více aktivit, využívat fragmenty. V takovém případě by jsme ale riskovali zavedení nových chyb a uživatelské testování by bylo nutné provést znovu.

7 Závěr

V této poslední kapitole bych rád předvedl postup přidání aplikace na trh Google play, poté uvedl možná budoucí rozšíření a nakonec zhodnotil průběh celé práce.

7.1 Publikování aplikace

V případě naší aplikace jsem se rozhodl o publikování na trhu Google Play. Oproti nabízeným variantám se jedná o nejpopulárnější možnost a Google svým vývojářům nabízí několik velmi nápomocných funkcí. Například statistiky stažení, celková charakteristika cílové populace, zabezpečení vydané aplikace a dokonce i tipy na vylepšení. Podívejme se nyní na celý průběh přidání.

7.1.1 Build aplikace

S buildem release verze aplikace nám hodně pomáhá samotné Android Studio, i když stejného výsledku by se dalo použít jen za pomocí gradlu. Android Studio nabízí možnost generování tzv. podepsaného balíčku, kde vytvořený .apk soubor je certifikovaný pomocí privátního klíče, který se dá jednoduše vyrobit. Tím docílíme důvěryhodnosti aplikace. Poté co máme vygenerovaný vlastní klíč je už potřeba jen správně konfigurovat build, kde má naše IDE opět předpřipravenu defaultní konfiguraci release.
Po vytvoření spustitelného balíčku aplikace se přesouváme do webové aplikace Google Play Console. K přihlášení je potřeba jen Google účet a vývojářský registrační poplatek. Pro přihlášení použiji školní google účet. Uvnitř aplikace nás nyní pro publikaci naší aplikace zajímají jen následující body

- Vydání aplikace
- Záznam v obchodu
- Hodnocení obsahu
- Cena a distribuce

Vydání aplikace

Tato funkce nabízí několik různých kanálů ve kterých můžeme vydávat. Nachází se tu kanál interního testování, uzavřený kanál - pro alfa testování, otevřený kanál - pro beta testování, a nakonec námi žádaný kanál produkční. Produkční kanál dává ostatním uživatelům Google Play přístup ke stáhnutí aplikace. Pro vytvoření vydání je nutné dodat podepsaný .apk balíček, název, verzi, a napsat patch notes vytvořené verze.

Záznam v obchodu

Součástí této sekce jsou informace a podklady k popisu aplikace na trhu Google Play. Jedná se tedy o název aplikace, její krátký a poté úplný popis pro různé druhy zobrazení a nakonec grafické prvky. Ke grafickým prvkům stačí logo 512x512 pixelů, které je použito jako spouštěcí ikona a poté potřebujeme ještě hlavní grafiku, což je obrázek v rozlišení 1024 x 500 použitý jako pozadí aplikace při zobrazení na trhu. Také se tu specifikují kategorie do kterých naše aplikace spadá.

Hodnocení obsahu

Hodnocení obsahu slouží k charakterizování věkových a národnostních skupin, pro které je aplikace bezpečná pro použití. K získání hodnocení stačí vyplnit krátký dotazník, který se zaměřuje na věci jako je obsah násilí, či nevhodných slov.

Cena a distribuce

A na závěr je nutné specifikovat do kterých států bude aplikace distribuována a za jakou cenu. Také se zde určuje zda aplikace obsahuje reklamy a je nutné udat souhlas s vývozními zákony USA a pokyny pro obsah systému Android. Poté co máme všechny povinné informace vyplněné, už je potřeba jen počkat na schválení aplikace do trhu. Schválení proběhlo v našem případě do několika hodin a aplikace je k dostání na tomto odkazu[1].

7.2 Možná budoucí rozšíření

V oblasti vývoje se málokdy dostaneme do situace, kdy by nebylo možné aplikaci nebo program nějak dále rozšířit či vylepšit. V průběhu této práce již bylo zmíněno několik možných vylepšení, ale bude vhodné je zmínit v závěru, poté co je znám kontext celé práce. Možná rozšíření jsou inspirována hlavně nápady vedoucího práce pana Balíka, který už měl koncept aplikace delší dobu rozmyšlený, ale také uživateli, kteří se zúčastnili testování a pomohli tak přinést nový nadhled nad řešený problém.
Jako hlavním problémem, i přes vyvíjení maximální snahy o zjednodušení práce se vstupy, pořád vidím zdlouhavost načítání potřebných informací, ať už se jedná o ukládání položek nebo o tvorby nákupních seznamů. Jako možné řešení bych viděl využití technologie Google Voice, která skrz léta vývoje dosáhla obrovského progresu. Google voice se nacházel i v konkurenčních aplikacích a tvorba nákupních seznamů s ním byla opravdu jednoduchá.
Dále by bylo vhodné nad vytvořenou aplikací vytvořit více pokročilých funkcí, jako je například chytré nabízení nákupních seznamů, které by podle frekvence spotřebovávání daných typů potravin nabízelo automaticky vygenerované seznamy, nebo přidat možnosti jako je vyplňování ceny nákupu a nad to například přidat funkci statistiky výdajů.
Dále také existuje možnost propojení s aplikacemi, která pracují s podobnými daty. Například pod vedením pana Balíka je paralelně vyvíjena aplikace na tvorbu receptů. Té by jsme mohli dodávat obsah z naší lednice a získat tak ideální recept, nebo alespoň inspiraci.
Jako poslední velké rozšíření bych rád zmínil server pro uchovávání načtených produktů. Každý uživatel momentálně musí projít zdlouhavým procesem přidávání čarových kódů k produktům, které přidává poprvé. V případě že by jsme měli nějaký server, který by si k sobě ukládal čárové kódy a s nimi spojené typy a kategorie produktů, mohli by jsme tomuto problému předejít. Samozřejmě by tím vznikli další problémy, jako je filtrování nevhodných vstupů, zabezpečení posílaných dat a deploy všech potřebných serverů. Proto ještě variantou pro tento přístup je předpřiravit množinu dat a deployovat tuto množinu rovnou s aplikací, kde by vzniknul problém s kontinuálním přidávání nových produktů. Oba přístupy jsou ale nadmíru složité a jedná se spíše o polemizaci možností.

[1] *FoodTracker - Obchod Google Play.* URL: `https://play.google.com/store/apps/details?id=cz.cvut.fel.foodtracker`.

7.3 Zhodnocení práce

Po úspěšném vývoji je čas rozebrat silné a slabé stránky námi vytvořené aplikace, jako ponaučení do budoucích projektů.
Osobně jsem nadmíru spokojený s backendem aplikace a tím myslím práci s databází a injikování dependencí do potřebných modulů. Myslím že vytvořený přístup je dostatečně robustní, aby postačil jako základ pro možná budoucí rozšíření, ale je i poměrně jednoduše napsaný. Takže v případě nutnosti změn do této části kódu by neměl být problém se rychle zorientovat a pokračovat ve vývoji podle připraveného vzoru.
Kde osobně vidím problém, je v uživatelském rozhraní. V uživatelském testování jsme sice získali poměrně kladnou odezvu, ale z osobní zkušenosti práce s jinými moderními aplikacemi si myslím, že je pořád co zlepšovat. Nebylo by od věci dodat nějakou vlastní grafiku, doplnit více aplikací a celkově aplikaci více zútulnit. Také vidím problém v čitelnosti a vůbec složitosti kódu controllerů, které se starají o zpracování dat z uživatelského rozhraní. Myslím že některé části se dali vyřešit lépe, například aktualizování listů po mazání prvků. Místo psaní vlastního listeneru, by stálo za to použít některý z listenerů přitomných v Android SDK, kterým by jsme určitě dosáhli podobného výsledku. Bohužel jsem si takovéto věci uvědomil až při dokončování vývoje a jsem si jistý že v případě vyvíjení nové aplikace by jsem již dokázal odvést lepší práci.

7.4 Závěr práce

V průběhu této práce jsme vytvořili aplikaci na práci s potravinami, která by v případě větší časové investice mohla mít potenciál na konkurování velkým značkám z tohoto odvětví. I v současném stavu si ale myslím že aplikace může být do spousty domácností šikovným pomocníkem a nejedná se tedy jen o pouhý základ na kterém by se eventuálně dalo stavět, ale o finální produkt, který má hodně možností kudy růst, jak bylo naznačeno v přechozí kapitole.
Bude obecně zajímavé sledovat situaci na námi cíleném trhu v budoucnosti, kdy se v domácnostech začínají objevovat i tzv “chytré” lednice. Je možné že takové lednice by mohli aplikaci našeho typu popostrčit výš v žebříčku populárnosti a třeba se i časem stanou podobné aplikace standardem pro práci s potravinami v domácnosti. Tímto průlomem ani nemusí být samotné lednice, ale třeba i nový standard čárových kódů, který by nějakým způsobem dokázal v kódu zachytit i další informace, díky kterým by jsme mohli produkty načítat mnohem rychleji. Co se týče Android aplikací, jsem přesvědčen že jejich dominance jen poroste a tvorba aplikací zůstane jako jednou z hlavních odvětví softwarového vývoje. Osobně jsem rád za zkušenosti a vědomosti získané v průběhu této práce a rád bych je využil i později na trhu práce.

Příloha A

Uživatelská příručka

I přesto že je FoodTracker mobilní aplikace a je tedy kladen důraz na jednoduchost a aby uživatel na první pohled porozuměl funkčnosti, je vhodné dodat k finálnímu produktu uživatelskou příručku, která na jednom místě shrne všechny informace pro potenciálního uživatele bez nutnosti číst celou bakalářskou práci.

A.1 Systémové požadavky

K používání aplikace je samozřejmostí nutnost telefonu Android, s minimální API úrovní 15. K instalaci nejsou nutná žádná specifická oprávnění a veškerý software by neměl přesáhnout velikost 10 MB.

A.2 Instalace aplikace

Aplikaci je možno instalovat buď vložením .apk souboru, který je přiložen na CD, do telefonu, a poté co je schválena instalace z cizího zdroje se o instalaci postará operační software. Druhou možností je stáhnout aplikaci z obchodu Google play na následujícím odkazu[1]. Pro odinstalování aplikace je nutné použít postup specifický pro vaše zařízení.

A.3 První kroky

Při prvním spuštění aplikace se dostáváte do rozcestníku, který vede do jednotlivých stěžejních funkcí. V této kapitole projdeme všechny tyto moduly a vysvětlíme si jejich funkce.

A.3.1 Přidávání položek

Přidávání položek obsahuje několik políček, které jsou nutná pro vyplnění, jinak není možné produkt uložit. Tyto políčka jsou název produktu, jeho kategorie a zakoupené množství. K množství je možné dodat jednotku, pro lepší přehlednost zásob a také je možné specifikovat cílové úložiště a jeho datum expirace. U produktů u kterých máme k dispozici čárový kód, je možný tento čárový kód načíst a daný produkt se nám uloží do databáze spolu s vyplněnými daty. Při dalším ukládání stejného produktu už tedy nemusíme vyplňovat název a kategorii produktu a ani použité jednotky. Lokace uložení je taktéž předvyplněna, ale ne pomocí čárového kódu, ale při prvním otevření aktivity je nalezeno úložiště s největším počtem prvků. Poté si aktivita pamatuje poslední použité úložiště dokud není zavřena.

[1] *FoodTracker - Obchod Google Play.* URL: `https://play.google.com/store/apps/details?id=cz.cvut.fel.foodtracker`.

A.3.2 Prohlížení obsahu lednice

Při prohlížení obsahu jsou uživateli zobrazeny jen kritické informace, s tím že informace o místě uložení je signalizována barvou. Bílé pozadí mají položky bez specifikovaného úložiště, další každé úložiště si pak bere jinou barvu. Uživatel by si ale měl dávat pozor na to že při použití víc jak 7 úložišť už budou všechna ostatní úložiště také bílá. Dále tu máme tlačítko které seřazuje položky tak, aby první byly ty které mají v nejbližší době vypršet, poté jsou položky bez specifikovaného data expirace a nakonec jsou zobrazeny již vypršelé produkty. Při kliknutí na jednotlivé položky se nám otevře dialog, kde je možné mazat položky, měnit jejich množství, nebo změnit přirazení kategorii.

A.3.3 Nákupní seznamy

Nákupní seznamy jsou rozděleny na 2 záložky. První záložkou je vytváření nákupního seznamu, kde v horním listu vidíme aktuální seznam, pod ním je hned textový vstup, kde může uživatel přidávat položky, které ještě nemá uložené v databázi. Pod textovým vstupem jsou pak jednotlivé kategorie, vytvořené při ukládání produktu a seznam 10 nejpopulárnějších položek.
V historii nákupních seznamů jsou seznamy seřazeny od nejnovějšího k nejstaršímu. Při kliknutí na seznam se otevře obrazovka, kde jsou vypsány všechny produkty ze seznamu a pomocí tlačítka plus je možné je rychle uložit do lednice. Po uložení se položka odškrtne. Dále je tu také možnost mazat položky z již vytvořeného seznamu pomocí obrázku křížku.

A.3.4 Nastavení

V nastavení je možné si zosobnit aplikaci, tak aby nás notifikovala v námi vybraný čas a s předem definovanou rezervou. Rezerva se vybírá pro jednotlivé kategorie v listu kategorií a čas se vybírá pomocí tlačítka choose time. Také je potřeba povolit zasílání notifikací odškrtnutím prvního políčka. Po změně konfigurací je nutné tyto změny uložit.

Příloha B

Programátorská příručka

Součástí práce je také příručka pro importování a práci s projektem, která je zaměřená pouze na Android studio. V případě využívání jiného IDE je nutné hledat pomoc jinde.

B.1 Importování projektu

Importování projektu provedeme v Android studio pomocí **New -> Open** a vyhledáme složku se zdrojovým kódem. Android studio by již mělo rozpoznat přítomnost gradle konfiguračního souboru a složka FoodTrackeru bude označená ikonkou Android Studio. V opačném případě je nutné otevřít složku FoodTracker a vybrat soubor build.gradle.

B.2 Spuštění projektu

Při importu projektu pravděpodobně bude Android Studio hlásit neznámé třídy. Je nutné tedy provést synchronizaci gradlu, kterou nám IDE samo nabídne. Poté je potřeba spustit **Build -> Make project**, pro vygenerování dagger souborů které v projektu injikují závislosti. Aplikaci pak můžeme spustit pomocí run buď na cílovém zařízení, nebo na emulátoru.

B.3 Strukturování souborů

Zdrojové soubory jsou rozděleny podle funkčnosti do několika složek. Velmi zjednodušeně se dají významy složek vysvětlit takto

- Activities - GUI aplikace, obsahuje podsložku dialogů a fragmentů které jsou používány k dosažení rezponzivnosti s uživatelem.
- Adapters - Cokoliv co rozšiřuje třídu adapter, což je třída která pomáhá seznamům pracovat s obsaženými daty.
- Codescanning - Třídy potřebné pro propojení projektu s Barcode Scannerem
- Controllers - Obsahuje controllery - třídy které se starají o zpracování dat z uživatelského rozhraní.
- DAO - V této složce jsou objekty rozhraní které spojuje datovou a servisní vrstvu.
- Definitions - Konstanty a třídy pro interní používání
- Entities - Datové entity
- Exceptions - Použité výjimky
- Helpers - Pomocné třídy pro práci s databází, nebo statické metody jako je např parsování textu
- Modules - Potřebné třídy pro běh dagger frameworku
- Services - Servisní vrstva databáze a procesy které běží na pozadí.

B.4 Logování

Pro rozběhání logování je nutné do resources přidat konfigurační soubor pro log4j. Tento soubor může být napsaný v XML,JSON,YAML nebo ve formátu properties. Konfigurace je také možná bez těchto souborů a to pomocí implementace ConfigurationFactory a Configuration. Existují ještě další způsoby, ale první dva jsou programátorsky nejčistší. Pro více info je vhodné použít přímo dokumentaci[1].

B.5 Potřebné knihovny

Díky gradlu se o většiny dependencí nemusí vývojář starat sám, ale i tak je nutné k rozběhání projektu manuální připravit některé knihovny. Potřeba je hlavně JDK Java 7 a vyšší. Dále specificky pro Android je potřeba Android SDK a Build tools. Pro naši aplikaci budou potřeba build tools 22.0.1 a Android SDK alespoň 5.1 nebo vyšší. V případě využití vyšších verzí je nutné modifikovat a ověřit funkčnost implementace.

[1] *Log4j dokumentace.* URL: `https://logging.apache.org/log4j/2.x/manual/configuration.html`.

Příloha C

Obsah CD

V přidaném CD jsou obsaženy následující složky

- text-source - Zdrojové soubory ke generování bakalářské práce.
- FoodTracker - složka s hlavní aplikací, která obsahuje potřebné konfigurační soubory a zdrojový kód.
- TestProjects - Testovací projekty, které byly vyrobeny před hlavní aplikací.
- text-final - finální verze textu bakalářské práce.

Literatura

[1] Linda Sui. *44% of World Population will Own Smartphones in 2017*. 21. pros. 2016. URL: https://www.strategyanalytics.com/strategy-analytics/blogs/smart-phones/2016/12/21/44-of-world-population-will-own-smartphones-in-2017 (cit. 10. 05. 2018).

[2] Jacob Poushter. *Smartphone Ownership and Internet Usage Continues to Climb in Emerging Economies*. 22. ún. 2016. URL: http://www.pewglobal.org/2016/02/22/smartphone-ownership-and-internet-usage-continues-to-climb-in-emerging-economies/ (cit. 10. 05. 2018).

[3] *Activity lifecycle*. URL: https://developer.android.com/reference/android/app/Activity (cit. 10. 05. 2018).

[4] Ľuboslav Lacko. "Aktivita a její životní cyklus". In: *Vývoj aplikací pro Android*. 1. vyd. Brno: Computer Press, 2015. Kap. 2, s. 65–71.

[5] *Context*. URL: https://developer.android.com/reference/android/content/Context (cit. 10. 05. 2018).

[6] *Fragments*. URL: https://developer.android.com/guide/components/fragments (cit. 10. 05. 2018).

[7] Ľuboslav Lacko. "Služby a broadcasty". In: *Vývoj aplikací pro Android*. 1. vyd. Brno: Computer Press, 2015. Kap. 15, s. 383–386.

[8] *Chytrý nákupní seznam Listonic*. URL: https://play.google.com/store/apps/details?id=com.l&hl=cs (cit. 10. 05. 2018).

[9] *Out of Milk - Grocery Shopping List*. URL: https://play.google.com/store/apps/details?id=com.capigami.outofmilk (cit. 10. 05. 2018).

[10] *Bring! Grocery Shopping List*. URL: https://play.google.com/store/apps/details?id=ch.publisheria.bring (cit. 10. 05. 2018).

[11] *Fridge, Foods, Expiration date*. URL: https://play.google.com/store/apps/details?id=jp.gr.java_conf.indoorcorgi.mykura (cit. 10. 05. 2018).

[12] *Best Before - Food Tracker*. URL: https://play.google.com/store/apps/details?id=com.peytu.bestbefore (cit. 10. 05. 2018).

[13] 2017. URL: http://www.mockingbot.com (cit. 27. 12. 2017).

[14] Dmitry Jemerov. *Hello World*. 19. čvc 2011. URL: https://blog.jetbrains.com/kotlin/2011/07/hello-world-2/ (cit. 10. 05. 2018).

[15] *Core J2EE Patterns,Data Access Object*. URL: http://www.oracle.com/technetwork/java/dataaccessobject-138824.html (cit. 10. 05. 2018).

[16] *Adaptive System Framework: A Way to a Simple Development of Adaptive Hypermedia Systems*. ADAPTIVE 2013, The Fifth International Conference on Adaptive and Self-Adaptive Systems and Applications. IARIA. Valencia,Spain, 2013, s. 20–25.

[17] *ZXing ("Zebra Crossing") barcode scanning library for Java, Android.* URL: `https://github.com/zxing/zxing` (cit. 10. 05. 2018).

[18] *Material design.* URL: `https://material.io/` (cit. 10. 05. 2018).

[19] *Pexels - Best free stock photos in one place.* URL: `https://www.pexels.com/`.

[20] Luboslav Lacko. "Notifikace, alarmy". In: *Vývoj aplikací pro Android*. 1. vyd. Brno: Computer Press, 2015. Kap. 6, s. 169–179.

[21] *FoodTracker - Obchod Google Play.* URL: `https://play.google.com/store/apps/details?id=cz.cvut.fel.foodtracker`.

[22] *Log4j dokumentace.* URL: `https://logging.apache.org/log4j/2.x/manual/configuration.html`.