ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA ELEKTROTECHNICKÁ

BAKALÁŘSKÁ PRÁCE

Mobilní aplikace pro správu skladu

Tomáš Arlt

Praha, 2018

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Arlt** Jméno: **Tomáš** Osobní číslo: **439590**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Nástroj pro inventuru skladu pro Android

Název bakalářské práce anglicky:

Inventory check tool for Android

Pokyny pro vypracování:

1) Proveďte analýzu existujících nástrojů pro evidenci zboží, prodej na bazaru a inventuru skladu.
2) Navrhněte a implementujte nástroj pro vkládání a správu zboží pro e-shop jako mobilní aplikaci pro systém Android.
3) Navrhněte vhodné funkce, mimo jiné vytvoření fotek produktu, úprava počtu kusů, umístění zboží ve skladu, přehledný výpis evidovaných produktů a synchronizace dat s e-shopem.
4) Vytvořenou 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: **19.02.2018** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: **30.09.2019**

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

Čestné prohlášení autora práce

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.

V Praze dne ..... ……

Podpis autora

Poděkování

Tímto bych chtěl poděkovat vedoucímu práce Ing. Martinu Balíkovi, Ph.D. za spolupráci, kontrolu a cenné rady při psaní a tvorbě mé bakalářské práce.

Anotace

Cílem práce bylo analyzovat a následně implementovat mobilní aplikaci pro systém Android. Aplikace je určená pro správu skladu a umožňuje provádět operace nad uskladněnými produkty včetně možnosti jejich rozšíření o dodatečné informace. Aplikace primárně slouží k evidenci zboží na skladě, bazaru nebo obchodě. Díky tomu slouží jako skvělý nástroj pro řízení inventur. Analytická část obsahuje porovnání existujících řešení, požadavky na funkcionalitu aplikace a diagramy reprezentující strukturu dat, scénáře použití a návrh uživatelského rozhraní. Další částí práce bylo vybrání technologií, které budou použity k implementaci aplikace. Vývoj probíhal v programovacím jazyce Java spolu s dalšími technologiemi potřebnými pro vytvoření úložiště dat, synchronizaci dat, předávání závislostí a testování. V kapitole o technologiích jsem vždy popsal více způsobů, jak dosáhnout požadovaného cíle. Na konci každé podkapitoly je obhájen výběr použité technologie. Podobným principem je vytvořena i další kapitola popisující implementaci aplikace. Zde jsou popsány důležité části aplikace a v případě, že je to možné, jsou uvedena i alternativní řešení. Ta mohou funkcionalitu vylepšit nebo optimalizovat její výkon. Následující dvě kapitoly pojednávají o testování aplikace, a to jak z hlediska testování její funkčnosti, tak i použitelnosti uživatelského rozhraní. Poslední kapitolou je shrnutí dosažených výsledků.

Klíčová slova

Správa skladu, evidence zboží, nástroj pro inventury, mobilní aplikace, Android, Java

Anotation

Goal of my bachelor thesis was to analyze and after implement a mobile phone application for Android System. Application is used for managing warehouses and enables to perform operations over stored goods, including options for expanding their information. Application's main purpose is to keep records about products in warehouses or shops. Because of this it can be used as a great tool for performing stocktaking. The analyze part consists of comparison existing solutions, definition of application's functionality, use cases and user interface prototype. Next part is about selection of technologies used for the development. Application was created in programming language Java with additional help of other technologies used for implementation of data storage, data synchronization, dependency management and testing. In this chapter are always described more ways how to achieve the desired goal. End of each subchapter contains an explanation why was particual technology used during development. With similar principle was made also the follow-up chapter describing implementation of the application. It provides information about creation of important parts and functionalities and if it is possible it also describes possible alternatives. Alternatives can implement functionalities in a better way or optimize their performance. Next two chapters are about testing the application. It describes methods of finding bugs preventing correct function of application and also mistakes of the user interface. The last chapter is summary of achieved goals.

Key words

Warehouse management, product record keeping, tool for stocktaking, mobile application, Android, Java

Obsah

1. Úvod

1.1. Využití aplikací pro správu skladů

V dnešní době se ruší čím dál více malých kamenných obchodů. Je to dáno důvodem, že lidé chodí častěji nakupovat do větších podniků jako je například Alza.cz. V těchto obchodech mají zákazníci větší výběr a nabídku služeb. Z tohoto důvodu se musí noví podnikatelé snažit vytvářet své podniky co největší. To přináší implementaci webového e-shopu, vznik nových poboček a také skladů, kde se zboží uschovává. Z tohoto důvodu musí existovat i aplikace, které podporují správu skladů. Aplikace mohou výrazně usnadnit práci například při vytváření faktur, inventurách, distribuci zboží, atd. Další výhodou je, že data jsou uloženy v digitální podobě. Tento typ dat se může zálohovat, a případně data ze zálohy obnovit, což výrazně snižuje riziko ztráty. V případě evidence pomocí papírových dokumentů by byla možnost zmizení dat výrazně vyšší. Informace o sortimentu a skladu jsou navíc dostupné a upravovatelné i z jiných lokalit, než je zboží uskladněno. Většinou je k tomu potřeba internetové připojení, ale to záleží na implementaci aplikace.

U menších kamenných prodejen se tyto nástroje dají také využívat. Většinou to však není potřeba. Majitelům to může přinášet zbytečnou práci navíc. Jelikož se produktů v obchodě tolik nenachází, odpadá jim nutnost vytvoření skladu, který by byl potřeba spravovat. Zboží je uskladněno většinou přímo v obchodě. V naší republice platí také zákon o elektronické evidenci tržeb. Prodejci tak mají povinnost vlastnit programy, které tento zákon dodržují. Díky tomu je programy ukládána spousta dat, které jsou pro majitele maloobchodu dostačující k tomu, aby si své zboží dokázali spravovat sami. Není tedy potřeba další program, který by vyžadoval vyplnění redundantních dat. Přesto v některých případech může být používání těchto aplikací prospěšné i v menších obchodech. Vybral bych však implementaci, která je přizpůsobená pro správu maloobchodu, ta může mít oproti čisté správě skladu jiné užitečné funkcionality.

Dalšími podniky, které mohou využívat aplikace pro správu skladů, jsou firmy, které se zabývají vyráběním produktů. Mezi ně mohou patřit i restaurace a fastfoody. Ke svému provozu potřebují materiály a další výrobky, které musí být někde uskladněny. Při výrobě se nemusí přesně vědět, kolik se spotřebuje materiálu na jeden kus výrobku. Při tvorbě produktu může dojít například k pochybení zaměstnance a nutnosti zahození materiálu. Proto nemůžeme nastavit automatické odečítání materiálů a vyplatí se využít některou aplikaci pro správu skladů. Většinou umožňují vytvářet nebo korigovat inventury, spravovat materiály a využívat spoustu dalších funkcionalit, které jsou navržené pro zlepšení bezpečnosti a efektivnosti administrace skladu. Na rozdíl od obchodů se budou tyto aplikace využívat spíše u menších podniků. Větší firmy vyžadují lepší funkcionalitu, minimální chybovost a podporu, jenž jim mohou nabídnout pouze nadnárodní korporace jako je například SAP.

1.2. Cíl práce

Cílem bakalářské práce je analýza a implementace mobilní aplikace pro skladové hospodářství. Hlavní využití bude při inventurách zboží na skladě. V případě, že se naleznou některé poškozené, přebývající nebo chybějící produkty, upraví se jejich počet přímo v aplikaci. Zároveň s aplikací bude realizován jiným studentem webový e-shop. Bude tedy vytvořen způsob synchronizace dat mezi oběma aplikacemi, aby nedocházelo k nekonzistenci dat. Budou navrženy a vytvořeny funkcionality, které uživateli ušetří čas a práci při evidenci uskladněného zboží. Aplikace bude poskytovat přehledné výčty a detaily uložených záznamů. Ve skladu nemusí pracovat pouze jeden zaměstnanec, proto bude k dispozici správa uživatelů a jejich autorizace.

2. Analýza

2.1. Analýza existujících řešení

Na trhu existuje spousta řešení, které mají obdobné funkcionality odpovídající zadání práce. Proto jsem některé takové aplikace nainstaloval na svém telefonu a zjistil, jak se liší a v čem jsou jedinečné. Vyzkoušel jsem poměrně velké množství aplikací. Díky tomu jsem získal inspiraci o tom, jaké by mohla aplikace obsahovat funkcionality a jak by mohl přibližně vypadat design. Dále jsme výsledek z analýzy existujících řešení prošli na konzultaci s vedoucím práce a upřesnili směr, kterým se bude implementace ubírat.

Následně budu vypisovat jednotlivé testované aplikace. Všechny jsou určené pro správu skladů nebo obchodů, mají tedy velkou řadu totožných funkcionalit. Proto se budu hlavně snažit popsat to, v čem je jednotlivá aplikace zajímavá a rozdílná oproti ostatním. V případě, že se některá funkcionalita opakuje, bude v popisu aplikace pouze zmíněna.

2.1.1. Stock and Inventory Simple

Stock and Inventory Simple[1] rozšířil záznam produktu o čárový kód. Zboží může být lehce identifikováno pomocí čtečky čárových kódů. Tato čtečka pro svou funkcionalitu využívá kameru mobilního telefonu a stačí pouze naskenovat čárový kód a zboží se automaticky přidá do faktury. Faktura může být tří typů: příchozí, odchozí a přesuny. Faktury se dají filtrovat pomocí intervalu dvou dat. Faktury se po vytvoření dají stáhnout do mobilu, odeslat e-mailem nebo nahrát na cloudové uložiště (Google Drive, Dropbox). Mezi některé další funkcionality, které stojí za zmínku, patří záloha a obnovení databáze nebo vytváření souhrnů o pohybech zboží, jeho prodeji a celkovém zisku.

Co se týče uživatelského rozhraní, tak mě zaujalo vyskakovací menu, které se zobrazí při stisku tlačítka pro přidání nového zboží. Následně si uživatel může vybrat, zda chce přidat produkt, kategorii nebo importovat/exportovat data. Myslím, že toto řešení je elegantní a ušetří místo na obrazovce. Při kliknutí na záznam o produktu se otevře editace. Při delším držení se zobrazí vyskakovací menu s dalšími možnostmi práce s vybraným produktem.

2.1.2. Dotykačka Sklad

Ke správné funkčnosti Dotykačka Sklad[2] musí být v telefonu nainstalována také aplikace Dotykačka EET. Toto omezení se nemusí některému uživateli zamlouvat. Mohl by chtít využívat pouze aplikaci pro správu skladu a přišlo by mu zbytečné si kvůli tomu kupovat obě aplikace po skončení zkušební lhůty. Aplikace oproti ostatním zavádí funkcionality pro správu dodavatelů. Dále se dají evidovat inventury a různé reporty.

V uživatelském rozhraní jsem našel chybu z hlediska použitelnosti. V případě, že chceme přidat nové zboží nebo kategorii musíme najít tuto možnost v levém vysouvacím menu. U výpisu produktů se nenachází žádné tlačítko na vytvoření nových záznamů. To může být pro prvotní uživatele velmi nepřehledné. Kategorie zboží jsou realizovány pomocí záložek.

[1] Stock and Inventory Simple - *https://play.google.com/store/apps/details?id=com.stockmanagment.next.app*
[2] Dotykačka Sklad - *https://play.google.com/store/apps/details?id=com.touchpo.android.stock*

2.1.3. Stock Controller - inventories

Stock Controller – inventories[3] umožňuje u zboží nastavit údaj, pomocí kterého bude systém oznamovat uživateli, že množství kleslo pod nastavenou minimální hodnotu. Nejedná se však o klasické notifikace, které jsou dostupné v jiných mobilních aplikacích. Pro tyto upozornění je vytvořená vlastní stránka, kde jsou vypsané informace o všech docházejících produktech. Počet kusů zboží se dá změnit pouze pomocí faktury. To zvyšuje bezpečnost, protože se eliminuje možnost, že by uživatel počet omylem upravil. Navíc se nestane, že by uživatel zapomněl vytvořit po změně počtu kusů k tomu odpovídající fakturu. Aplikace umožňuje přidávání fotek k produktům vyfocením nebo nahráním obrázku z galerie mobilního telefonu. Kamera také podporuje načítání čárového kódu, ale pouze pro získání jeho číselné reprezentace. To se využívá při vyplňování formuláře pro vytváření nových záznamů.

2.1.4. Inventory Management

Aplikace Inventory Management[4] také podporuje upozornění při klesnutí počtu kusů produktu pod nastavený limit. Umožňuje importovat a exportovat některá data prostřednictvím csv souborů. Autor tvrdí, že se aplikace velmi snadno používá, což je pravda v případě, že chceme pouze základní funkcionalitu, potřebnou pro správu skladů.

Design aplikace je dělán ve stejném duchu, jako autor zamýšlel použití aplikace. Tudíž velmi jednoduchý a nezáživný. V některých částech mi přišel nedodělaný.

2.1.5. My Business

My Business[5] rozšiřuje produkt je rozšířen o jednotku, čehož se dá využít v případě, že se zboží prodává například podle hmotnosti nebo objemu. Aplikace umožňuje vytvářet faktury, u kterých se dá nastavit sleva a upřesnit způsob úhrady. Po dokončení se automaticky vytvoří PDF dokument. Ten je následně po volbě uživatele odeslán nebo zobrazen. Systém umožňuje evidovat externí výdaje a přínosy spojené s provozem obchodu. Šance ztráty dat je redukována zálohováním a získáním souhlasu od uživatele před spouštěním akcí, které mění data. Dodavatelé a zákazníci mohou být importováni ze seznamu kontaktů v mobilním telefonu. Je dostupný i vlastní kalendář.

[3] Stock Controller – inventories - *https://play.google.com/store/apps/details?id=com.XNRSisbi.stockcontroller*
[4] Inventory Management - *https://play.google.com/store/apps/details?id=ankit.inventory.ankitarora.inventorymanagementsimple*
[5] My Business - *https://play.google.com/store/apps/details?id=com.segb_d3v3l0p.minegocio*

2.2. Porovnání aplikací

	Stock and Inventory Simple	Dotykačka sklad	Stock Controller - inventories	Inventory Management	My Business	**Cool storage**[6]
Pouze skladové hospodářství	✓	X	✓	✓	X	✓
Správa faktur	✓	✓	✓	✓	✓	**X**
Správa kategorií	✓	✓	✓	✓	✓	✓
Správa dodavatelů	X	✓	X	X	✓	✓
Čtečka čárových kódů	✓	X	✓	X	✓	✓
Export dat	✓	X	✓	✓	✓	✓
Import dat	✓	X	X	✓	X	✓
Notifikace nedostatku zboží	X	X	✓	✓	✓	✓
Přidávání fotek ke zboží	✓	X	✓	X	✓	✓
Statistiky / reporty	✓	✓	✓	✓	✓	**X**
Vytváření a načítání záloh	✓	X	X	X	✓	**X**
Dostupná bezplatná verze aplikace	✓	X	✓	✓	✓	✓

Tabulka 2.1: Porovnání existujících aplikací

2.3. Závěr analýzy existujících řešení

Skoro u každé zkoušené aplikace se daly nalézt nějaké drobnosti, které snižovali kvalitu a efektivitu práce se skladem. Nejvíce se mi líbila aplikace My Business. Měla pěkný design a také spoustu zajímavých funkcionalit, na které ostatní aplikace ani nepomyslely. My Business je však převážně určený pro správu maloobchodů, tudíž se zde nacházejí části, které se v běžném skladovém hospodářství nevyužijí.

V tomto odstavci shrnu všechny zajímavé nálezy získané procházením jednotlivých aplikací. Většina z nich bude do mé práce naimplementována.

[6] Cool storage – jméno výsledné aplikace bakalářské práce.

- Čtečka čárových kódů – u zboží bude možné přidat čárový kód pomocí číselné reprezentace nebo čtečky. Aplikace bude podporovat čtecí mód. Pomocí čtečky se načte kód zboží a aplikace vypíše o produktu všechny informace.
- Zobrazení vyskakovacího menu s možnostmi, po kliknutí na tlačítko pro přidání.
- Nastavení upozornění ke zboží, které bude kontrolovat, zda počet kusů neklesl pod minimum. Přehledný výpis těchto upozornění bude na samostatné stránce.
- K produktu bude možné pořídit fotografie pomocí kamery mobilního telefonu nebo nahrát obrázky přímo z Galerie.
- U zboží je přidána informace o jednotce, pod kterou se zboží distribuuje.
- Potvrzení, že uživatel chce opravdu provést akci, která způsobí změnu dat.
- Export a případně i import dat.

2.4. Business požadavky

Po používání již implementovaných řešení skladového hospodářství a konzultacích s vedoucím práce jsem se rozhodl nadefinovat některé požadavky na aplikaci. U každého požadavku je upřesněna jeho priorita. V případě, že má požadavek nastavenou vysokou prioritu, jedná se o požadavek potřebný pro správné fungování aplikace. Na druhou stranu požadavky s nízkou prioritou nejsou pro základní funkčnost aplikace potřebné a pouze poskytují uživateli nějaké nadstandardní možnosti práce se skladem nebo aplikací.

2.4.1. Nefunkční požadavky

Android API (Vysoká)

Aplikace bude vytvořená pro mobilní telefony se systémem Android [1]. Minimální podporovaná verze je systém, odpovídající definici API 15. Aplikace bude zkompilována a otestována na API 25.

Datové úložiště (Vysoká)

Aplikace bude využívat jako základní datové úložiště lokální databázi SQLite [2] uloženou v paměti mobilního telefonu. Databázová struktura uložiště bude vytvořena pomocí anotací z frameworku OrmLite [3].

Autorizace a autentizace uživatele (Vysoká)

Aplikace bude zabezpečená pomocí přihlašování. K využívání aplikace se bude uživatel autentizovat pomocí přidělených přihlašovacích údajů.

2.4.2. Funkční požadavky

Autorizace (Vysoká)

Aplikace umožní nepřihlášenému uživateli se přihlásit do aplikace pomocí svých přihlašovacích údajů. Aplikace vyžaduje autorizaci pouze při prvotním přihlášení nebo změně účtu.

Editace profilu zaměstnance (Vysoká)

Aplikace umožní uživateli editaci všech informací ve vlastního účtu.

Vytvoření profilu zaměstnance (Vysoká)

Aplikace umožní uživateli vytvořit účet s vygenerovanými přihlašovacími údaji pro nového zaměstnance.

Správa zboží (Vysoká)
Aplikace umožní uživateli provádět základní CRUD operace na položkách zboží. Aplikace umožní uživateli přidat ke zboží parametry, které budou definovány pouze pro vytvořené zboží.

Přidání čárového kódu ke zboží (Vysoká)
Aplikace umožní uživateli přidat k vybranému zboží čárový kód. Ten je zapsán číselnou reprezentací nebo je načten pomocí kamery a čtečky čárových kódů.

Správa kategorií (Vysoká)
Aplikace umožní uživateli provádět základní CRUD operace na položkách kategorie. Aplikace umožní uživateli při vytváření nové kategorie definovat parametry, které jsou pro všechny produkty z kategorie totožné. Aplikace umožní uživateli vytvářet podkategorie.

Výpis zboží (Vysoká)
Aplikace umožní uživateli vypsat informace o dostupném zboží na skladě. Aplikace umožňuje výpis jako detail nebo seznam položek.

Načítání zboží (Vysoká)
Aplikace umožní uživateli načíst detail zboží po naskenování čárového kódu. Aplikace umožní skenování pomocí kamery a čtečky čárových kódů.

Přidávání oznámení (Střední)
Aplikace umožní uživateli nastavit k vybranému zboží limit, od kterého se bude vytvářet a vypisovat notifikace.

Procházení oznámení (Střední)
Aplikace umožní uživateli procházet oznámení o docházejících zásobách na skladě.

Přidávání fotky ke zboží (Střední)
Aplikace umožní uživateli přidat ke zboží obrázky z Galerie mobilního telefonu nebo přidat fotografie získané pomocí kamery mobilního telefonu.

Přidání umístění zboží ve skladu (Střední)
Aplikace umožní uživateli přidat k vybranému zboží informaci o tom, kde se zboží ve skladu nachází.

Přidání výrobce zboží (Střední)
Aplikace umožní uživateli přidat k vybranému zboží výrobce.

Správa výrobců (Střední)
Aplikace umožní uživateli provádět základní CRUD operace na položkách výrobce.

Smazání profilu zaměstnance (Střední)
Aplikace umožní uživateli si smazat svůj vlastní profil.

Filtrování seznamu zboží (Střední)
Aplikace umožní uživateli filtrovat seznam zboží pomocí parametrů.

Synchronizace dat (Střední)
Aplikace umožní synchronizovat data s webovou aplikací e-shopu. Aplikace bude podporovat synchronizaci v obou směrech.

Exportování dat (Střední)

Aplikace umožní uživateli exportovat informace o zboží na skladě do csv souboru.

Logování akcí uživatele (Střední)

Aplikace bude ukládat informace do souboru o tom, jaké akce uživatel v aplikaci provádí. Logy budou sloužit k dohledávání informací v případě pochybení například ze strany zaměstnance.

Generování čárového kódu (Nízká)

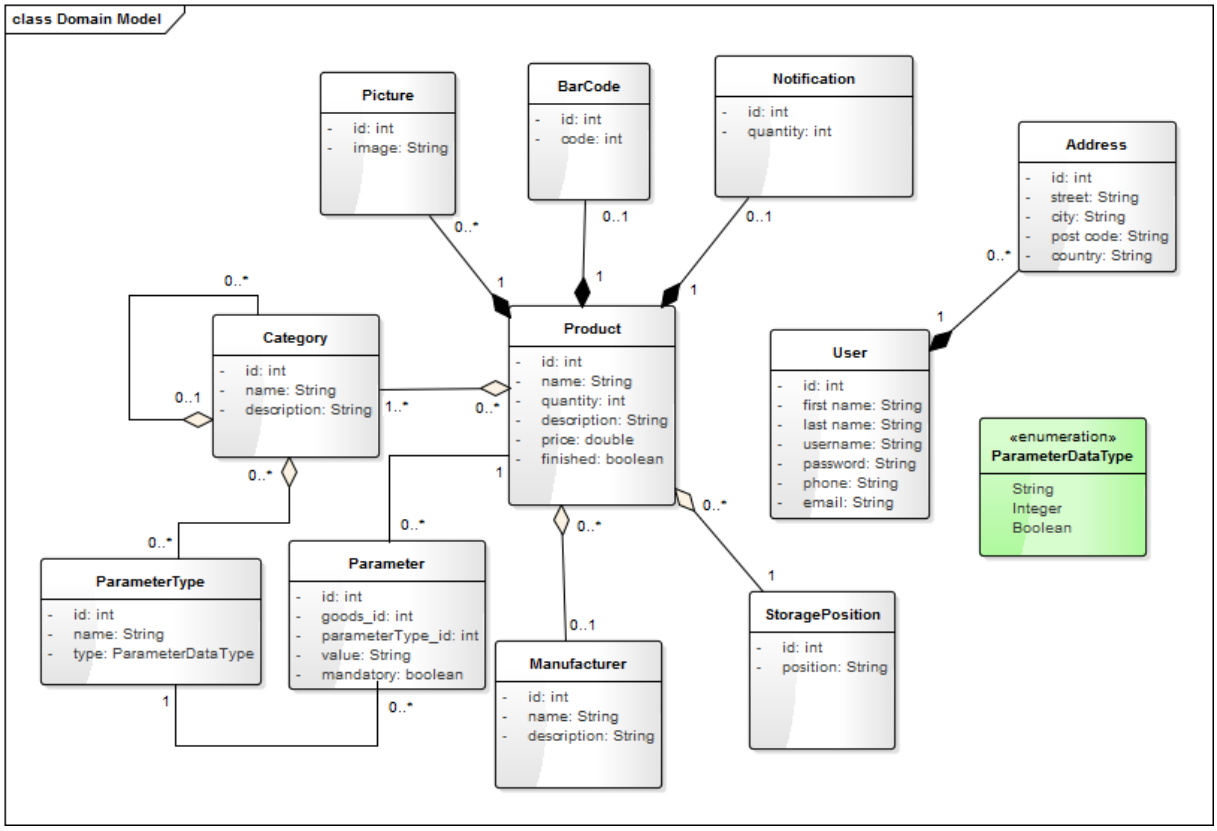
Aplikace umožní uživateli vygenerovat čárové kódy pro vybrané zboží.

Importování dat (Nízká)

Aplikace umožní uživateli import nového zboží pomocí souboru csv.

2.5. Domain model

Na začátku analýzy jsem vytvořil domain model aplikace. Model jsem postupně upravoval a rozšiřoval o nové třídy. Úpravy probíhaly na základě vymyšlení nových funkcionalit a úpravy stávajících. V následující kapitole se nachází popis jednotlivých tříd, jejich vlastností a vztahů s jinými třídami.



Obrázek 2.1: Domain model

2.5.1. Popis tříd

Jednou z nejzásadnějších tříd v celém modelu je entita pro reprezentaci zboží (Product). Mimo základních informací jako je jméno, popis a počet kusů jsou na zboží navázány i další třídy, které nám umožňují evidovat v záznamu o produktu doplňující data. Další entitou, která se může vytvořit a navázat na zboží je třída Notification. Maximálně jedna tato třída může být propojena s vybranou instancí zboží. Úkolem třídy je nastavit limit počtu kusů zboží ve skladu, při jehož dosažení

aplikace vytvoří upomínku na docházející zboží. K produktu se také dají přiřadit obrázky (Picture), čárový kód (BarCode), výrobce (Manufacturer) a lokace umístění zboží ve skladu (StoragePosition).

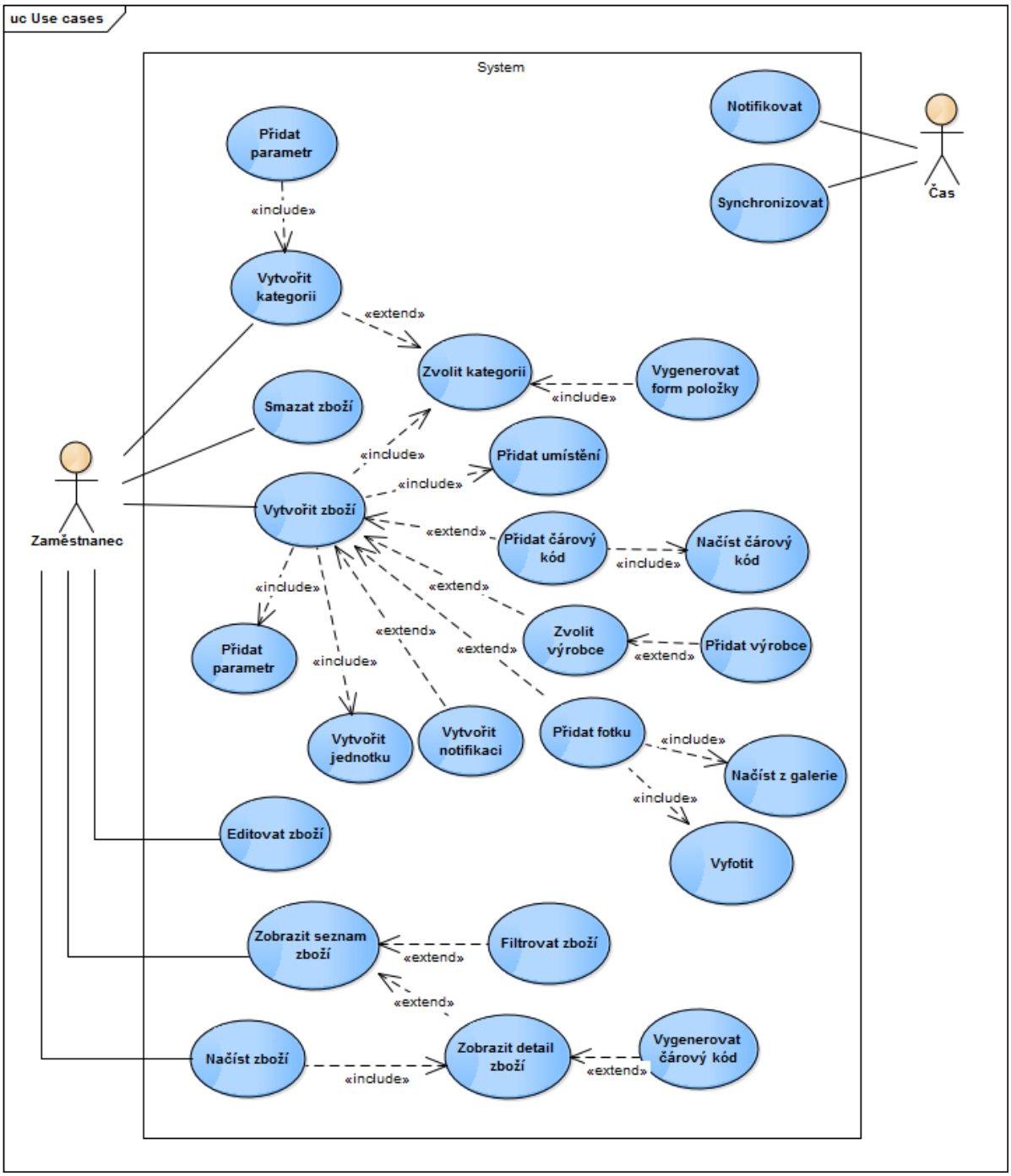
Další důležitou třídou je kategorie (Category). Interpretace kategorie v modelu se často měnila, jelikož se v průběhu analýzy vymýšlely nové funkcionality a upravovaly stávající. V prvotním návrhu byla pouze entita kategorie, která v sobě obsahovala žádnou až N entit zboží. Následující změnou bylo, že jsme do modelu přidali možnost vytváření podkategorie k již vytvořené kategorii. Dále se mi naskytl zajímavý nápad, že produkty ve stejné kategorie mohou mít některé společné parametry. Zboží by se mohlo pomocí těchto nových informací snadněji vyhledávat a filtrovat. Uživateli by byla také usnadněna práce při vytváření nového záznamu o zboží. Jelikož by se automaticky vytvořily formulářové prvky na základě vybrané kategorie a uživateli by stačilo pouze vyplnit jednotlivé hodnoty parametrů.

Díky tomu jsem se rozhodl, že tento způsob meta modelování poskytne zajímavé vylepšení mé aplikace. Nejdříve jsem tedy provedl analýzu, abych se dozvěděl jak je tato funkcionalita implementována v jiných, již existujících aplikacích. Analýzu jsem se rozhodl provést na třech větších českých e-shopech (Alza, Aukro a Obi). Pomocí analýzy jsem zjistil, jakým způsobem parametry fungují. Následovalo tedy přidání nových tříd pro parametry do domain modelu a jejich provázání s entitami zboží a kategorie. K správnému vytvoření parametru jsou potřeba dvě třídy. Jedna z nich (ParameterType) určuje, jakého je druhá třída (Parameter) datového typu. Datový typ je omezen hodnotami enumerátoru (ParameterDataType). Třída Parameter je následně navázána na zboží a obsahuje v sobě hodnotu parametru. U parametru je také evidována jeho povinnost. Jelikož chceme také ke zboží přidávat nové parametry, bez ohledu na zvolenou kategorii, musí být vztah mezi entitami Category a ParameterType parciální (nepovinný) na obou stranách.

Nesmíme zapomenout, že v aplikaci budeme chtít evidovat zaměstnance. K tomu slouží třída User, která obsahuje základní informace o zaměstnanci, mimo jiné jeho přihlašovací údaje. Heslo nebude v databázi uloženo v čitelné podobě, ale bude zabezpečeno pomocí hashovací funkce. Poslední používanou entitou je třída Address, která určuje adresu bydliště zaměstnanců.

2.6. Model případů užití

Po vytvoření business požadavků, byl čas na modelování případů užití aplikace. Kvůli přehlednosti byly případy užití rozděleny do dvou částí. Některé případy užití se vyskytují v obou modelech. V případě, že use case bude mít v obou případech stejný scénář užití, bude scénář vypsán pouze jednou. Mým snažením bylo vypsat scénáře pro nejdůležitější případy užití.



Obrázek 2.2: První část diagramu případů užití

2.6.1. Vytvořit zboží

Aplikace umožní zaměstnanci do skladu vytvořit záznam o novém zboží.

Scénář:

1. Zaměstnanec chce vložit do skladu záznam o novém zboží.
2. Aplikace zobrazí formulář pro přidání nového zboží.
3. Zaměstnanec stiskne tlačítko pro přidání fotografie.
4. Aplikace dá zaměstnanci na výběr, zda chce nahrát fotku z Galerie nebo vyfotit novou pomocí kamery.
5. Zaměstnanec vybere možnost pro vyfocení nové fotografie a pořídí nový snímek produktu.
6. Aplikace se vrátí zpět na formulář a uloží fotku.
7. Zaměstnanec vyplní potřebné údaje ve formuláři a klikne na tlačítko uložit.
8. Aplikace provede kontrolu zadaných formulářových dat.
   IF validated == true
   THEN Aplikace skočí na krok 9.
   ELSE Aplikace skočí na krok 2. a označí červeně chybně zadané hodnoty.
9. Aplikace uloží nové zboží do databáze a zeptá se uživatele, zda chce pokračovat ve vyplňování doplňujících informací o zboží.
10. Zaměstnanec zvolí, že chce vyplnit doplňující údaje.
11. Aplikace vygeneruje nový formulář pro doplňující informace.
12. Zaměstnanec začne vyplňovat údaje, nevyhovuje mu žádná kategorie a rozhodne se vytvořit novou. Spuštění alternativního scénáře: Vytvořit kategorii.
13. Zaměstnanec klikne na tlačítko pro načtení čárového kódu pomocí čtečky.
14. Aplikace spustí čtečku čárových kódů.
15. Zaměstnanec namíří kameru na čárový kód a vyčká, než čtečka kód načte.
16. Aplikace se vrátí zpět na formulář a vloží číselnou reprezentaci čárového kódu do formulářového prvku.
17. Zaměstnanec vyplní zbylé údaje a klikne na tlačítko další.
18. Aplikace provede validaci formulářových dat.
    IF validated == true
    THEN Aplikace skočí na krok 19.
    ELSE Aplikace skočí na krok 11. a označí červeně chybně zadané hodnoty.
19. Aplikace vygeneruje nový formulář pro vyplnění parametrů vzhledem k vybrané kategorii.
20. Zaměstnanec vyplní vygenerované parametry a klikne na tlačítko uložit.
21. Aplikace provede validaci formulářových dat.
    IF validated == true
    THEN Aplikace skočí na krok 22.
    ELSE Aplikace skočí na krok 19. a označí červeně chybně zadané hodnoty.
22. Aplikace uloží doplňující informace k již vytvořené entitě zboží.

Alternativní scénáře:

2.6.2. Vytvořit kategorii

Aplikace umožní přidat uživateli novou kategorii pro zboží. V tomto případě, je scénář vypsán pro přidávání kategorie v průběhu vytvoření zboží. Scénář popisující vytvoření nové kategorie v základním menu bude však velmi podobný. Z tohoto důvodu už znovu vypsán nebude.

1. Zaměstnanec se rozhodne vytvořit pro zboží novou kategorii a stiskne tlačítko pro vytvoření nové kategorie.
2. Aplikace se přesune na stránku s formulářem pro vytvoření nové kategorie.
3. Zaměstnanec vyplní základní údaje (název a popis). Zaměstnanec chce ke kategorii přidat dva parametry, klikne tedy na šipku u nápisu Parametry.
4. Aplikace rozbalí sekci Parametry, kde se nachází formulář pro vytvoření jednoho parametru.
5. Zaměstnanec vyplní údaje pro první parametr a klikne na tlačítko Nový parametr.
6. Aplikace vygeneruje do sekce další nové formulářové prvky pro přidání dalšího parametru.
7. Zaměstnanec vyplní nově vygenerované formulářové prvky a stiskne tlačítko Přidat.
8. Aplikace provede validaci formulářových dat.
   IF validated == true
   THEN Aplikace skočí na krok 9.
   ELSE Aplikace skočí na krok 6. a označí červeně chybně zadané hodnoty.
9. Aplikace uloží nový záznam o kategorii do databáze a vrátí se zpět do formuláře pro tvorbu nového zboží. Automaticky je formulářový prvek nastaven na vytvořenou kategorii. Ve scénáři se pokračuje na místě, kde se skončilo (krok 13).

2.6.3. Zobrazit detail zboží

Aplikace umožní zaměstnanci zobrazit informace o vytvořeném zboží na skladě.

1. Zaměstnanec se rozhodne zjistit informace o zboží. V některém z menu klikne na tlačítko Zboží.
2. Aplikace zobrazí seznam dostupného zboží na skladě včetně rozdělení do kategorií.
3. Zaměstnanec omezí počet vypsaného zboží pomocí filtrů.
4. Aplikace modifikuje seznam vypsaného zboží podle zadaných filtrů.
5. Zaměstnanec klikne na záznam o hledaném produktu.
6. Aplikace otevře stránku s detailem zboží.

2.6.4. Načíst zboží

Aplikace umožní zobrazit zaměstnanci detail produktu po načtení čárového kódu.

1. Zaměstnanec zná čárový kód produktu a potřebuje zjistit o zboží bližší informace. V některém z menu klikne na tlačítko Načíst zboží.
2. Aplikace spustí čtečku čárových kódů.
3. Zaměstnanec namíří kameru na čárový kód a vyčká, než čtečka kód načte.
4. Aplikace zkontroluje, zda je v databázi zboží s načteným čárovým kódem.
   IF nalezeno == true
   THEN Aplikace otevře stránku s detailem zboží.
   ELSE Aplikace zobrazí hlášku o tom, že zboží není k dispozici. A vrátí se na předchozí stránku.

2.6.5. Smazat zboží

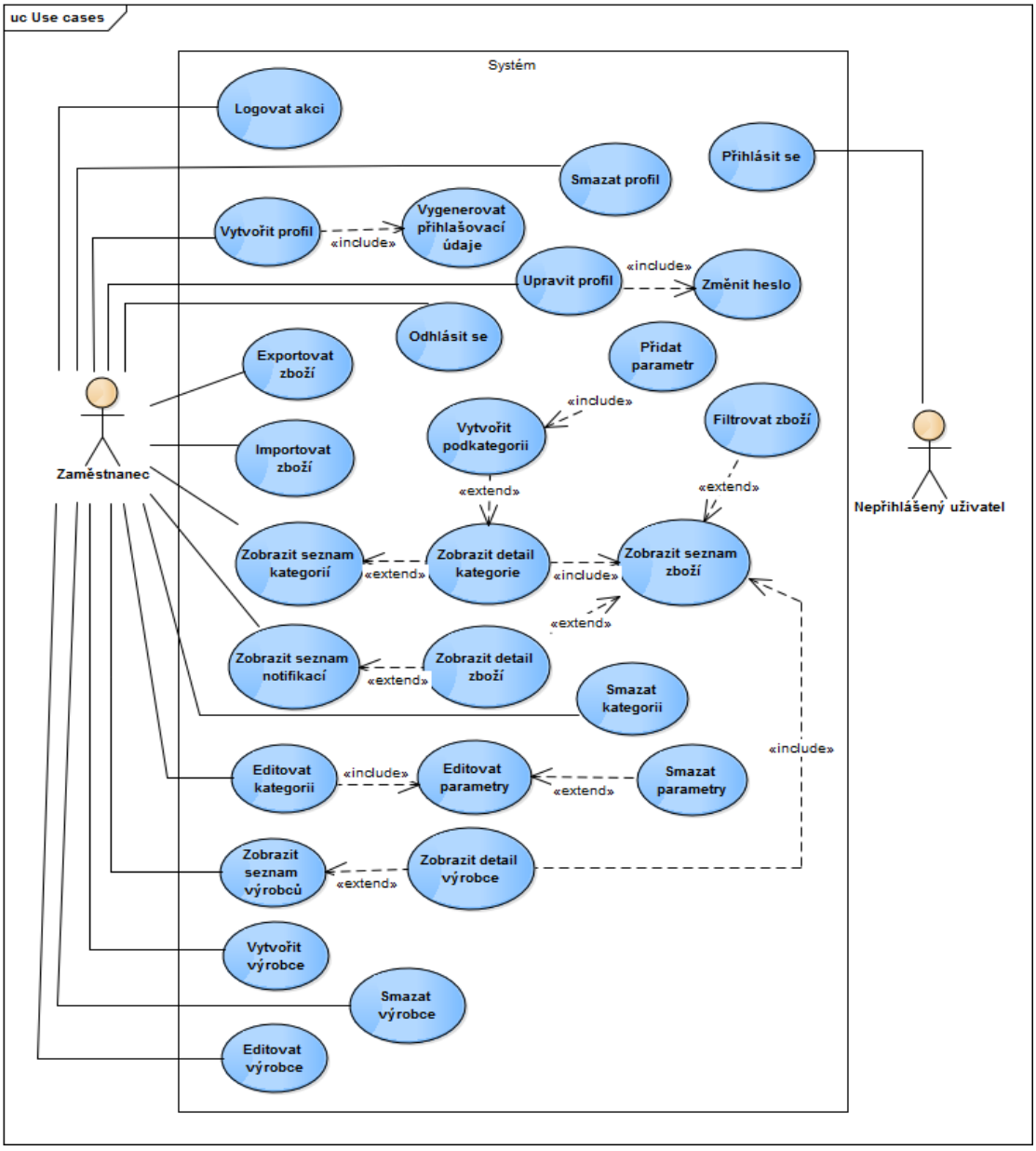
Aplikace umožní zaměstnanci smazat záznam o zboží.

1. Zaměstnanec se rozhodne smazat záznam o zboží z databáze. V některém z menu klikne na tlačítko Zboží.
2. Aplikace zobrazí seznam dostupného zboží na skladě včetně rozdělení do kategorií.
3. Zaměstnanec vyhledá zboží, které chce smazat a klikne na tři tečky na konci řádku záznamu.
4. Aplikace otevře vyskakovací menu s dalšími možnostmi manipulace a práce se zbožím.
5. Zaměstnanec klikne na tlačítko smazat.

6. Aplikace vytvoří vyskakovací okno, ve kterém se zeptá uživatele, zda chce provést tuto akci.
7. Zaměstnanec zvolí, zda chce s akcí pokračovat nebo jí chce zrušit.
8. IF potvrzeno = true
   THEN Aplikace vymaže všechny entity spojené se zbožím včetně entity zboží z databáze. Následně se vrátí na seznam zboží.
   ELSE Aplikace nic neprovede a zůstane na seznamu zboží.

2.6.6. Editovat zboží

Aplikace umožní uživateli editovat informace o zboží. Umožňuje nastavit a případně odebrat od zboží parametry, notifikace, … V podstatě umožňuje nastavovat a měnit vše co je dostupné při tvorbě zboží. Scénář pro tento případ užití, je podobný jako pro případ Vytvořit zboží. S tím rozdílem, že jsou předvyplněné ve formuláři některé informace. Z tohoto důvodu není scénář vypsán.



Obrázek 2.3: Druhá část diagramu případů užití

2.6.7. Vytvořit profil

1. Zaměstnanec se rozhodne vytvořit profil pro nového zaměstnance. Otevře Nastavení z některého menu. Zde následně otevře stránku pro Správu uživatelů.
2. Aplikace vykreslí stránku pro Správu uživatelů.
3. Zaměstnanec klikne na tlačítko pro vytvoření nového uživatele.
4. Aplikace vygeneruje přihlašovací údaje pro nového zaměstnance a zobrazí je.
5. Zaměstnanec předá informace pro přihlášení novému zaměstnanci.

2.6.8. Upravit profil

1. Zaměstnanec se rozhodne změnit své přihlašovací údaje a doplnit zbývající informace. Z některého z menu otevře stránku Nastavení. Zde následně otevře stránku pro Správu uživatelů.
2. Aplikace vykreslí stránku pro Správu uživatelů.
3. Zaměstnanec klikne na tlačítko Upravit profil.
4. Aplikace zobrazí formulář pro změnu údajů.
5. Zaměstnanec změní své přihlašovací údaje a vyplní zbylé informace. Nakonec klikne na tlačítko Uložit.
6. Aplikace edituje informace o zaměstnanci v databázi a zobrazí stránku pro správu uživatelů.

2.6.9. Vytvořit podkategorii

1. Zaměstnanec se rozhodne vytvořit k již vytvořené kategorii podkategorii. Otevře stránku Zboží a kategorie z některého menu.
2. Aplikace zobrazí seznam dostupného zboží na skladě včetně rozdělení do kategorií.
3. Zaměstnanec otevře kategorii, do které chce přidat podkategorii.
4. Aplikace vykreslí detail o kategorii se seznamem zboží v kategorii.
5. Zaměstnanec klikne na tlačítko pro vytvoření nového prvku.
6. Aplikace vykreslí pop-up menu.
7. Zaměstnanec klikne na tlačítko Vytvořit kategorii z nového vykresleného menu.
8. Aplikace vykreslí formulář pro vytvoření nové kategorie a pokračuje se dále podle případu užití Vytvořit kategorii.

2.6.10. Zobrazit seznam notifikací

1. Zaměstnanec chce vidět, jaké zboží na skladě dochází. V některém z menu klikne na tlačítko Upozornění.
2. Aplikace vykreslí stránku s upozorněními, kde se nachází záznamy o veškerém docházejícím zboží na skladě.
3. Zaměstnanec si může prohlédnout seznam a případně otevřít ze seznamu detail zboží.

2.7. Návrh uživatelského rozhraní

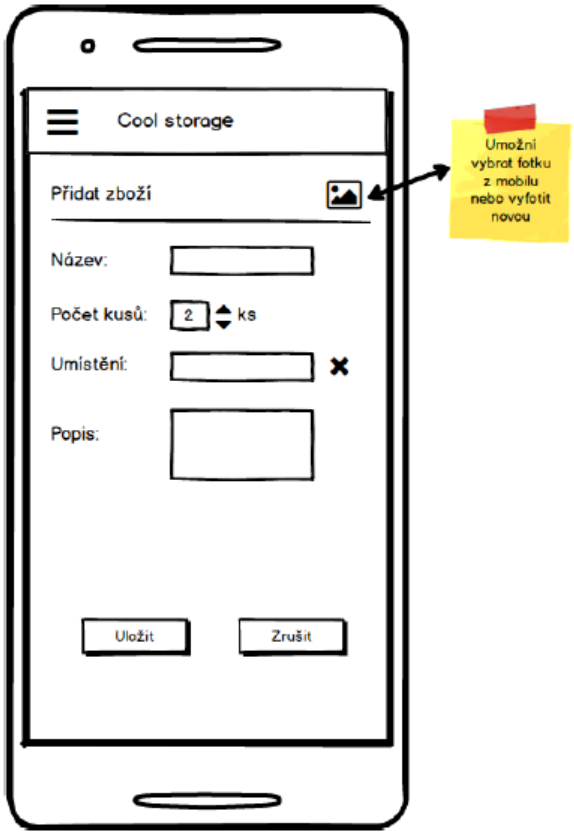
V této části textu bude popsána další část analýzy, konkrétně návrh uživatelského rozhraní. Tento návrh byl vytvořen v programu Balsamiq mockups [4]. Jedná se o tzv. low fidelity prototype, což je typ prototypu uživatelského rozhraní. Tento typ se vyznačuje svou jednoduchostí a nebere v potaz přesné umístění prvků dle norem, barvy prvků, atd. Ve vývoji aplikací se používá pro prvotní návrh, který je předán zákazníkovi. Dodavatel softwaru tak ušetří spoustu prostředků a času v případě, že by zákazník byl s uživatelským rozhraním nespokojený a chtěl ho předělat.

Kvůli stručnosti bakalářské práce jsem se rozhodl přidat pouze obrázky jednotlivých obrazovek bez popisků fungování jednotlivých prvků. Popis nejzásadnějších prvků byl proveden přímo v prototypu pomocí poznámek. V případě částí popisující funkcionality jsou obrazovky seřazeny v pořadí průchodu. Návrh není kompletní, v dokumentu se nenachází část pro nastavení a správu uživatelů.

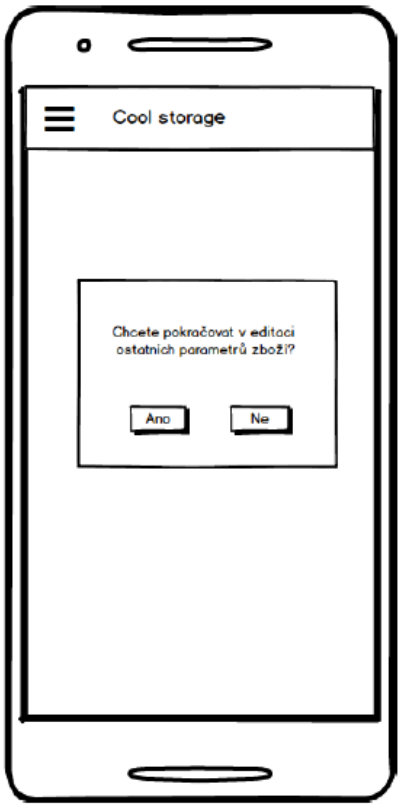
2.7.1. Základní části aplikace

Obrázky 2.4, 2.5, 2.6: Přihlašovací stránka, vysouvací menu aplikace, hlavní stránka aplikace

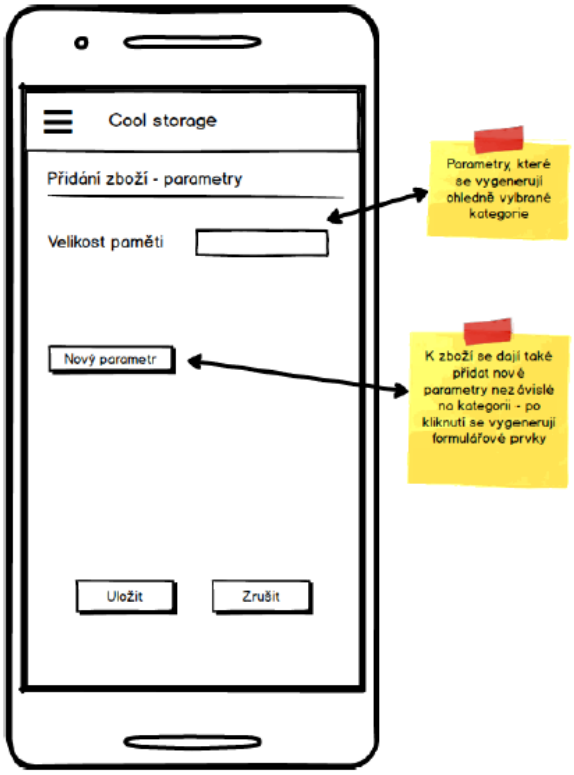
2.7.2. Přidání zboží



Obrázky 2.7, 2.8 – Vyskakovací menu přidávání zboží, formulář pro základní vytvoření produktu

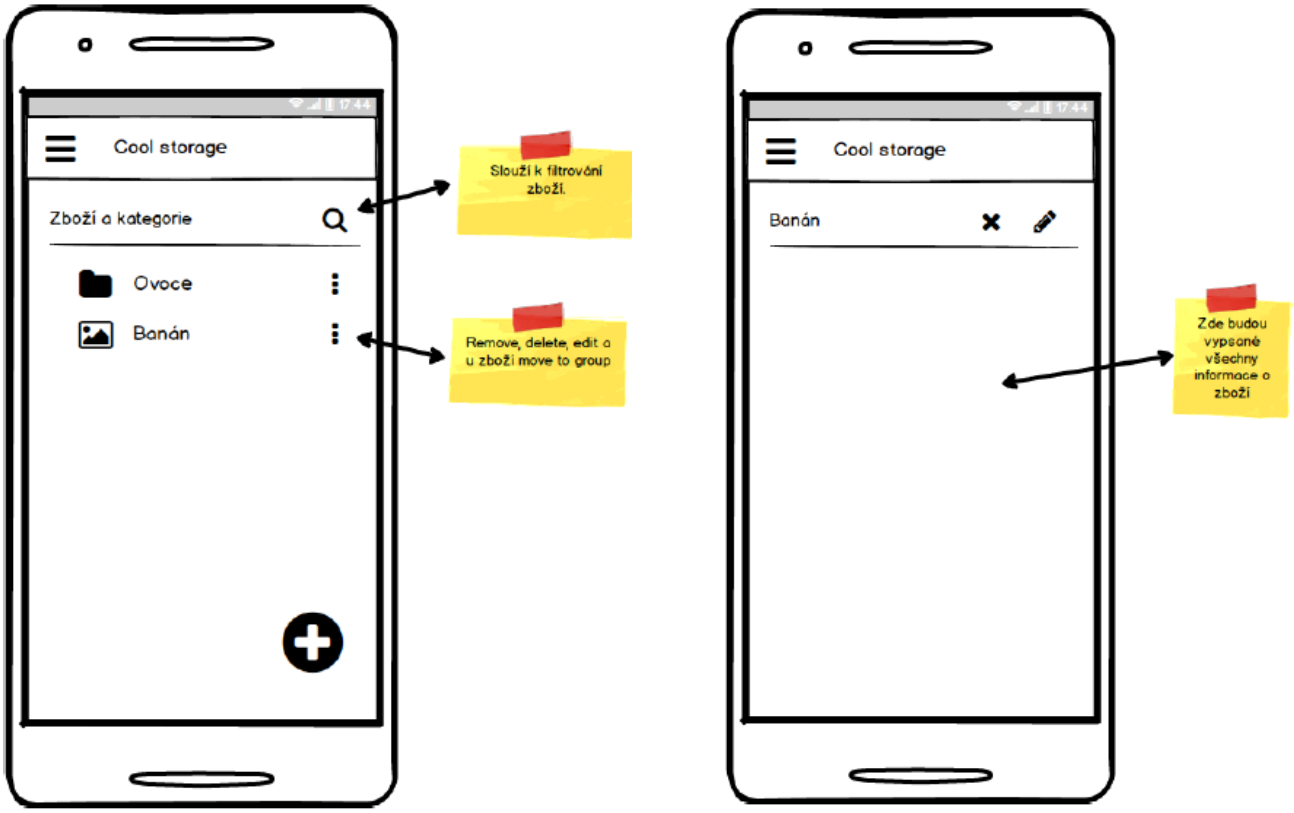


Obrázky 2.9, 2.10: Zjištění dalšího postupu, formulář pro dodatečné informace o produktu



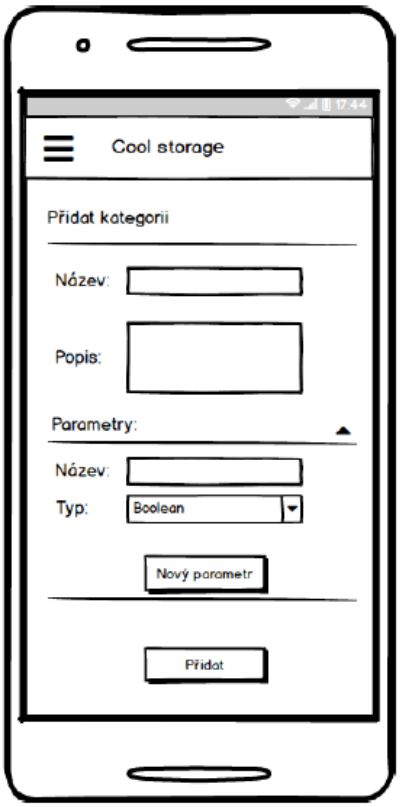
Obrázek 2.11: Formulář s vygenerovanými parametry dle kategorie

2.7.3. Zobrazení zboží



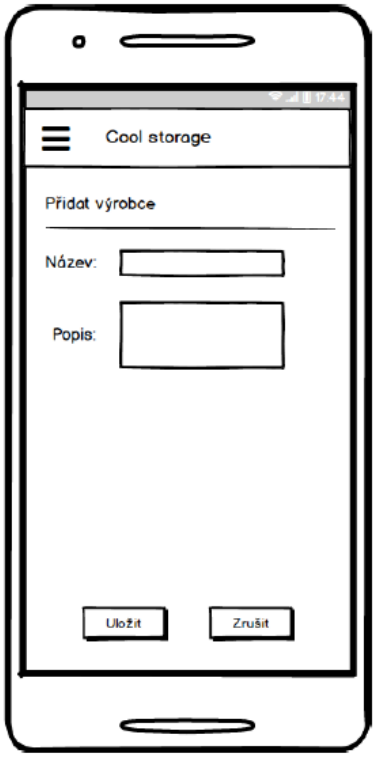
Obrázky 2.12, 2.13: Seznam zboží a kategorií, detail zboží

2.7.4. Přidání kategorie



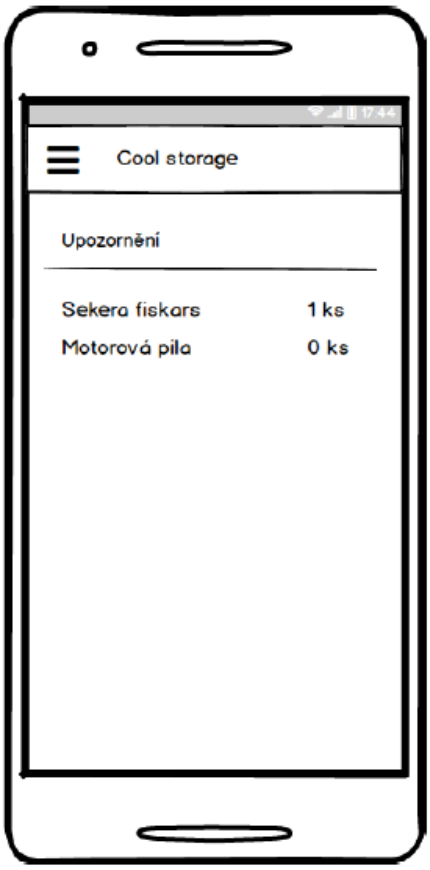
Obrázek 2.14: Formulář pro vytvoření nové kategorie

2.7.5. Správa výrobců



Obrázky 2.15, 2.16: Přehled výrobců, formulář pro vytvoření výrobce

2.7.6. Zobrazení upozornění



Obrázek 2.17: Stránka s upozorněními na docházející zboží

3. Použité technologie

V této kapitole jsou popsány použité technologie v aplikaci. Protože je možné vybrat si z velkého množství jazyků, knihoven a frameworků, je u jednotlivých podkapitol popsáno více možností, jak dosáhnout požadovaného cíle.

3.1. Typ aplikace

Existuje více možností, jak vytvořit mobilní aplikaci. Každý způsob vytvoření má své výhody či nevýhody. Je na vývojáři, aby vybral řešení, které nejlépe odpovídá požadavkům získaným z analýzy a konzultace s klientem.

3.1.1. Webová aplikace

Jedná se o webovou aplikaci s responzivním designem, který prvky stránky přesouvá a upravuje dle rozlišení obrazovky zařízení. K vytvoření webové aplikace se používají standardní technologie pro tvorbu webů, např. JavaScript, CSS, HTML5, PHP a spousta dalších. Největší výhodou tohoto řešení je dostupnost aplikace ze všech zařízeních, které disponují webovým prohlížečem a připojením k internetu. Technologie na vytváření webů nejsou přizpůsobené pro vytváření mobilních aplikací. Proto přistupování k zařízením a senzorům mobilního telefonu může vyžadovat využití speciálních knihoven nebo není z bezpečnostního hlediska možné.

3.1.2. Hybridní aplikace

Tento typ aplikace se využívá v případech, kdy je potřeba dostupnost aplikace pro více platforem a nedisponuje s dostatečnými prostředky pro vytvoření aplikací pro jednotlivé systémy. Při vývoji hybridních aplikací se používají frameworky, které zařizují komunikaci s nativním prostředím platformy. Mezi nejznámější frameworky patří Ionic, React Native, PhoneGap a další. Tímto přístupem docílíme rychlého vývoje aplikace s větším využitím. Ztratíme však na výkonu aplikace díky režii a abstrakci, kterou přináší použití frameworku.

3.1.3. Nativní aplikace

Vývoj nativní aplikace je zaměřen pouze na určený typ mobilního systému. Využívají se tak optimalizované knihovny a funkcionality, což přináší maximální výkon aplikace a možnost snadné integrace senzorů a zařízení mobilního telefonu do aplikace. Další výhodou je existence konvencí a pravidel pro implementaci funkcionalit a uživatelského rozhraní aplikace. Při respektování těchto předpisů bude výsledná aplikace pro budoucí spotřebitele co nejpřívětivější a bude obsahovat věci, na které jsou zvyklí.

3.1.4. Závěr

Po prozkoumání jednotlivých možností jsem se rozhodl pro výběr nativní aplikace. Toto rozhodnutí bylo učiněno z následujících důvodů. Základní funkcionalita aplikace nesmí být závislá na internetovém připojení, jelikož ve skladech může být připojení omezené. Vybraná platforma aplikace je pouze systém Android. Nativní aplikace umožňuje rychlý a jednoduchý přístup k zařízením mobilního telefonu, jako je například kamera.

3.2. Programovací jazyk

3.2.1. Kotlin

Kotlin [5] byl vývojáři Androidu nedávno prohlášen za další oficiální jazyk pro tvorbu mobilních aplikací. Tento programovací jazyk má své silné stránky v bezpečnosti a redukci napsaného množství kódu. Kotlin chrání uživatele především před NullPointerExeptions,[7] pomocí jejich detekce na úrovni kompilace programu. Kotlin také umožňuje uživateli v kódu použít JavaScript nebo jakoukoliv knihovnu dostupnou v JVM.[8] Kotlin s JVM zaručuje stoprocentní kompatibilitu.

3.2.2. Java

Java je dalším oficiálním programovacím jazykem pro vývoj aplikací pro Android. Jedná se o jeden z nejvíce používaných programovacích jazyků. Je oblíbený kvůli objektovému přístupu, bezpečnosti, správě paměti a výkonnosti. Díky velké rozšířenosti, době existence jazyka a masivní komunitě programátorů existuje spousta veřejně dostupných návodů a již vytvořených, optimalizovaných frameworků a knihoven. Správné využití těchto zdrojů usnadní vývojářům spoustu práce a umožní dosáhnout co nejlepších výsledků. Java je vyvíjena jako open source,[9] to poskytuje uživateli možnost porozumění a kontroly nad jeho implementací.

3.2.3. Závěr

Pro implementaci aplikace jsem se rozhodl využít programovací jazyk Java. Důvodem byly mé dlouhodobé zkušenosti nejen v programování mobilních aplikací, ale také desktopových nebo webových Enterprise aplikací.[10] Další výhodou je možnost využití již zmiňovaných návodů, frameworků a knihoven.

3.3. Android API

Android API [6] je rozhraní, které uživateli dodává přístup k poskytovaným zdrojům a funkcionalitám systému Android. S technologickým pokrokem vývoje mobilních aplikací dochází ke zlepšení, vytvoření nových funkcionalit. Funkcionality jsou navrženy a optimalizovány pro určitou verzi systému Android. Je potřeba mezi sebou odlišit jednotlivá rozhraní, aby docházelo ke správnému volání funkcionalit systému. Rozlišení je vytvořeno přiřazením unikátního čísla, tzv. API level, které definuje jednotlivé verze rozhraní systému Android.

Vývojář je povinen v konfiguraci aplikace upřesnit množinu rozhraní využitých v aplikaci. Jedním z atributů je určení minimálního API, kterým musí systém mobilního telefonu disponovat. Není-li toto omezení splněno, systém zařízení zamezí instalaci aplikace. Dalším atributem je compileSdkVersion, jenž určuje rozhraní systému Android, pro které je aplikace kompilována. Využívané funkcionality jsou tak získávány primárně z nastaveného rozhraní nebo nižších v případě nutnosti. Vývojáři je umožněno deklarovat targetSdkVersion, což definuje API, pod kterým je aplikace testována. Správná funkčnost aplikace však není na tomto záznamu závislá. Dodává pouze Androidu dodatečné informace, které systém využívá při používání aplikace, vedoucí k optimalizaci aplikace a správnému využívání prostředků jednotlivých verzí systému. Posledním atributem je nastavení maximálního

[7] NullPointerExeptions – výjimka nastávající v případě, že používáme proměnnou s nepřiřazenou hodnotou.
[8] Java Virtual Machine – virtuální stroj, který zpracovává zdrojové kódy napsané např. v jazyce Java.
[9] Open source – licence, která umožnuje vývojáři procházet, modifikovat a využívat zdrojové kódy.
[10] Enterprise aplikace – aplikace určená například pro organizace. Obsahuje spoustu funkcionalit, které jsou dosaženy využitím velkého množství použitých technologií.

podporovaného Android API. Aplikace tak nemůže být nainstalována na mobilní telefon, který disponuje vyšším než maximálně nastaveným rozhraním. Toto omezení se však nedoporučuje nastavovat, protože je potřeba dosáhnout funkčnosti aplikace pro co největší množství zařízení.

Android poskytuje Support Library [7], což je technologie, která umožňuje pro vývoj aplikací využití více rozhraní a vytváří kompatibilitu mezi jejich jednotlivými funkcionalitami. Za předpokladu, že je pro danou funkcionalitu a API, využívané systémem mobilního zařízení, implementována kompatibilita v Support Library, může vývojář používat funkcionalitu vyšších rozhraní v systémech s nižším rozhraním a naopak. Android spolu se Support Library pro systémy různých verzí automaticky vloží kód zaručující kompatibilitu nebo nahradí použitou funkcionalitu obdobně fungující z nižšího rozhraní. Aplikace je tak ve většině případů nezávislá na verzi systému mobilního zařízení. Problém nastane pouze v případě, že není implementována kompatibilita v Support Library. Na takové případy však upozorňuje vývojové prostředí Androidu vzhledem k atributům definujícím použitá rozhraní v konfiguraci aplikace.

Po důkladné analýze jednotlivých Android rozhraní byla pro implementovanou aplikaci vybrána následující rozhraní. Minimální rozhraní je nastaveno na API 15, kompilované a target na API 25.

3.4. Datové úložiště

V aplikaci je potřeba místo pro ukládání dat. Pro účel aplikace postačí relační databáze. Databáze může být umístěna lokálně na zařízení nebo na serveru. Aplikace požaduje základní funkcionalitu bez internetového připojení, tudíž můžeme databázi na serveru zamítnout.

3.4.1. SQLite

SQLite je relační databáze napsaná v Jazyce C. Mezi její hlavní výhody patří: absence konfigurace databáze, existující transakce a obecnost databáze. To umožňuje provozovat databázi na kterémkoliv operačním systému. Jedná se pouze o lokální databázi. Data se ukládají přímo na disk do jediného vytvořeného souboru. V případě používání čisté SQLite databáze by se musely tabulky a dotazy vytvářet ručně pomocí SQL jazyka. Což by mohlo prodloužit práci a vytvořit potencionální hrozbu v bezpečnosti kvůli SQL injection.[11] Musela by být naimplementována opatření, která by hrozbu eliminovala. Oproti tomu frameworky většinou obsahují již vytvořená opatření nebo poskytují nástroje k jejich vytvoření.

3.4.2. ORMLite

Framework ORMLite funguje na principu Object-relation mapping. Tento princip využívá objektového návrhu entit, které reprezentují databázové tabulky. Převod na databázové tabulky je definován pomocí anotací. Framework obsahuje své vlastní anotace, pomocí kterých se dají vytvořit jednotlivé sloupce tabulky včetně primárních a cizích klíčů a vztahů mezi tabulkami. Podporovány jsou všechny základní datové typy včetně speciálního pro uložení celého objektu. Objekt, který implementuje rozhraní java.io.Serizable je pak v době ukládání serializován na sekvenci bytů, která je vložena do databáze. V případě získávání dat objektu z databáze se byty deserializují a vytvoří uložený objekt. Mimo základní anotace podporuje také framework JPA [8] anotace. JPA anotace mají výhodu v tom, že jsou známější a umožňují přenést datový model do jiné aplikace, která nepoužívá ORMLite.

[11] SQL injection – typ útoku na databázovou vrstvu aplikace.

```java
@DatabaseTable(tableName = "picture")
public class Picture extends AbstractEntity {

    @DatabaseField(canBeNull = false)
    private String image;

    private Bitmap bitmapImage;

    @DatabaseField(foreign = true, foreignAutoRefresh = true, columnName = "product_id")
    private Product product;

    public String getImage() { return image; }

    public void setImage(String image) { this.image = image; }

    public Product getProduct() { return product; }

    public void setProduct(Product product) { this.product = product; }

    public Bitmap getBitmapImage() { return bitmapImage; }

    public void setBitmapImage(Bitmap bitmapImage) { this.bitmapImage = bitmapImage; }

    @Override
    public String getClassName() { return "Picture"; }
}
```

Obrázek 3.1: Příklad databázové tabulky vytvořené pomocí ORMLite anotací.

ORMLite podporuje větší množství databází než jen SQLite. Mezi nejznámější patří MySQL, Postgres, Microsoft SQL Server a H2. Databázová podpora není pouze jediná silná stránka frameworku. Další výhodou je přítomnost tříd DAO[12] a možnost jednoduše vytvářet databázové dotazy pomocí k tomu určeného stavitele.

3.4.3. Firebase

Firebase [9] není pouze dalším řešením databáze a vytváření její struktury. Jedná se o celou řadu služeb poskytovaných firmou Google. Mimo databázi se dají využít služby pro reportování a sledování chyb, ukládání souborů v cloudu a spoustu dalších. Databáze pracuje na technologii NoSQL a je umístěná v cloudu. To dělá databázové operace velmi rychlé a umožňuje přístup až milionům uživatelů najednou, bez ztráty výkonu. Data jsou uchovávány ve formátu JSON a synchronizována v reálném čase. V případě, že mobilní telefon nemá dostupné internetové připojení, nejsou data synchronizována a jsou ukládána pouze lokálně. K synchronizaci dojde znovu po obnovení připojení. Databáze automaticky vyřeší konflikty, které nastaly opožděnou synchronizací. Firebase je vynikající službou na řešení problému ukládání dat. Google poskytuje bezplatnou verzi s omezenými parametry, které jsou dostačující pro aplikace menších rozměrů.

3.4.4. Závěr

Jako úložiště dat jsem si vybral SQLite databázi, protože se jedná o skvělou a jednoduchou technologii na vytváření lokálních databází pro mobilní aplikace. Kvůli ulehčení generování SQL příkazů a jejich navazování s příslušnými objekty aplikace jsem se rozhodl databázovou strukturu generovat pomocí anotací z frameworku ORMLite. Nabízela by se možnost použití JPA anotací, ale jejich podpora je v ORMLite nedostačující. Neumožňují nastavení automatického načítání připojených objektů

[12] DAO – data access object – třída pro manipulaci s daty v databázi.

k výsledkům databázových dotazů. Vývojář musí získat objekty pomocí cizího klíče a vytvořit nový databázový dotaz, což je v některých případech složité.

3.5. Dependency injection

Dependency injection [10] je návrhový vzor, který je často přítomen v komplexních aplikačních frameworcích. Pracuje na principu inversion of control, který využívá k řízení a předávání dat třídám speciálně navržené služby. Dependency injection využívá k předávání závislostí následující způsoby: vložení pomocí parametru v konstruktoru, metody pro nastavení argumentů nebo využití metod vybraného rozhraní. Dependency injection usnadňuje vývojáři práci při vytváření a distribuci tříd s vysokou znovu-použitelností a návrhovým vzorem Singleton.[13]

3.5.1. RoboGuice

RoboGuice [11] vkládá potřebné závislosti do zdrojového kódu pomocí reflexe. Využití reflexe způsobuje výrazné zpomalení využívaných funkcí. Umožňuje předávat vysokou škálu typů Android závislostí. Například zdroje z Android balíčků drawable a resources, instance třídy View, systémové služby nebo libovolné objekty. Framework využívá k vymezení prostoru a vybrání závislosti své vlastní anotace.

3.5.2. Butter Knife

Framework Butter Knife [12] nepoužívá pomalé reflexe. Místo nich vytváří rozšířené třídy s přidaným automaticky vygenerovaným kódem, který zajišťuje správné přiřazení závislostí. Toto řešení zatěžuje zařízení jen v průběhu kompilace. Kód se generuje na základě anotací frameworku. Mezi zajímavé funkcionality frameworku patří přidávání závislostí do posluchačů na kliknutí a adaptérů určených pro reprezentaci a výpis dat.

3.5.3. Dagger

Dagger [13] je framework, který k vytváření dependence injection využívá anotace. Anotace jsou zpracovány v době kompilace projektu. Toto řešení je velmi rychlé a nezatěžuje zařízení tolik jako frameworky, které využívají reflexi. Závislosti jsou vytvářeny a uchovávány na jednom místě, které se nazývá Object graph. Odtud jsou následně jednotlivé instance modulů předávány ostatním. Závislosti jsou rozděleny do logicky oddělených modulů. Moduly jsou následně sjednoceny do komponenty, která poskytuje aplikaci interface pro získávání závislostí. Dagger se dá kromě Android aplikací také využít pro programy napsané v jazyce Java.

3.5.4. Závěr

Android ve svém základním balíčku nepodporuje vytvoření dependency injection a je potřeba využít framework k tomu určený. Já jsem z vypsaných možností vybral framework Dagger, z důvodu jeho oblíbenosti, jednoduchosti a rychlosti.

3.6. Synchronizace dat

Aplikace podporuje výměnu dat mezi webovým e-shopem a aplikací. Synchronizace musí být umožněna oběma směry. Je to dáno tím, že změna zboží může být inicializována ze strany e-shopu nebo pomocí mobilní aplikace.

[13] Singleton – návrhový vzor, zaručující existenci pouze jediné instance třídy.

3.6.1. Společná databáze

Nejjednodušším řešením synchronizace dat by bylo využití společné databáze pro obě aplikace. Databáze by nemohla být uložena lokálně, ale musela by být umístěna na serveru. Z toho vyplývá nevýhoda, že by mobilní aplikace ke své funkčnosti vyžadovala internetové připojení. Tato nevýhoda by se dala obejít využitím již zmiňované databázové služby Firebase, která data synchronizuje automaticky a umožňuje používat databázi i lokálně. Nevýhodou tohoto řešení by byla závislost aplikací na společné databázi, která by musela být správně navržena, aby nedocházelo k chybám při manipulaci s daty. Oddělené databáze přináší jednodušší modifikaci databáze a systému v případě, že by došlo k rozšíření některé z aplikací.

3.6.2. REST API

REST API [14] je způsob komunikace a přenosu dat mezi aplikacemi. REST umožňuje využívat ke tvorbě komunikace již existující standardizované komunikační protokoly. Nejpoužívanějším protokolem pro tvorbu REST rozhraní se stal protokol HTTP, který je používán pro komunikaci na internetu. Díky tomu nejsou pro vytvoření API potřebné žádné přídavné knihovny či balíčky. REST může vytvářet end-pointy pro vytvoření, získání, editaci a mazání (CRUD) libovolně dostupných dat. Získaná data z rozhraní jsou dostupná ve více formátech (XML, JSON, YAML). Přesný typ je určen pomocí HTTP požadavku. Správně implementované REST API by mělo splňovat šest vlastností a omezení, jež jsou popsány níže.

a) Client-Server – toto omezení popisuje nezávislost klienta a serveru. To znamená, že úprava dat v mobilní aplikaci nesmí vést k modifikaci dat na serveru a naopak.

b) Stateless – jedná se o vlastnost, která specifikuje, že jednotlivě spuštěné požadavky na rozhraní by měly být nezávislé. Dále by jednotlivá volání měla obsahovat všechny potřebná data pro dokončení, REST by neměl využívat pro získání odpovědi lokálně uložených dat na straně serveru.

c) Cache – pro omezení počtu volání rozhraní, by měl REST navádět uživatele k ukládání již získaných dat lokálně do aplikační cache.[14] To se zařídí přidáním atributů do odpovědi. Atributy určují možnost, lokálního uložení dat a případnou dobu vypršení jejich platnosti.

d) Uniform interface – samostatné rozhraní nesmí být závislé na modelu, službách a dalších částech aplikace, pod kterou se REST API nachází.

e) Layered system – rozhraní by se mělo skládat z více vrstev. Každá vrstva má svou vlastní funkcionalitu a účel. Komunikace musí probíhat vždy pouze mezi sousedními vrstvami.

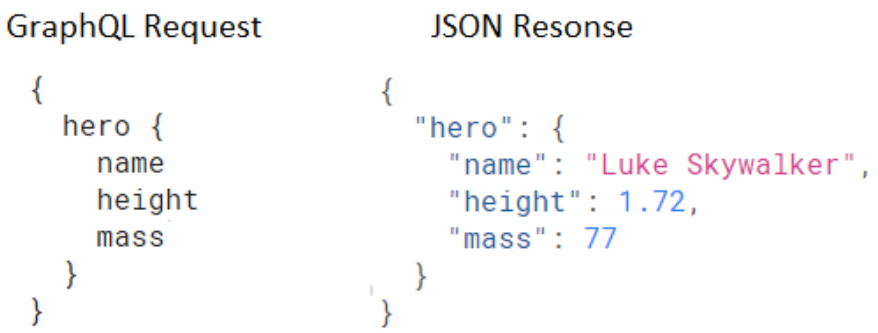
f) Code on demand – jediná nepovinná vlastnost. Umožňuje pomocí rozhraní přenášet části kódu. Kvůli možnosti využití při implementaci rozhraní a aplikací rozdílných programovacích jazyků se tato vlastnost často nevyužívá. Mohlo by dojít k narušení bezpečnosti.

3.6.3. GraphQL

GraphQL [15] je dotazovací jazyk určený pro získávání dat z rozhraní. Pro získávání odpovědi se neprovádějí na serveru žádné akce, jako tomu je u jiných API. Klient odešle na API dotaz, ve kterém specifikuje data, která chce získat. Rozhraní mu následně pošle odpověď s příslušnými daty. Na rozdíl od REST API, kde se pro získání některých dat musí provést více volání za sebou, dokáže klient získat

[14] Cache – hardwarová/softwarová struktura pro ukládání a následné získávání dat.

všechna data pomocí jediného GraphQL požadavku. Díky eliminaci serveru jsou dotazy prováděny velmi rychle. To dodává dobré výkony i na starších zařízeních. GraphQL není specificky navržený pro určitou platformu, a tak může být použit na klientech všech platforem.



Obrázek 3.2: Příklad získávání dat z API pomocí GraphQL požadavku.

Konfigurace API spočívá v identifikaci dostupných objektů a jejich atributů. Po správném nastavení a vytvoření podpůrných metod bude umožněno klientovi zažádat o příslušná data.

```
type Query {
  me: User
}

type User {
  id: ID
  name: String
}
```

Obrázek 3.3: Definice dostupných dat pomocí API.

3.6.4. Závěr

Pro synchronizaci dat jsem se rozhodl využít REST API. V tomto případě by se více hodilo GraphQL, ale kvůli nedostatku času a zkušeností s jeho používáním jsem se rozhodl pro mnou známé REST API.

4. Implementace

V této kapitole jsou popsány nejdůležitější a nejzajímavější naimplementované funkcionality mobilní aplikace. V některých podkapitolách mohou být popsány i alternativní možnosti řešení problému, které by problém mohly zdokonalit nebo vyřešit lepším způsobem.

4.1. Formuláře

Jednou z nejdůležitějších částí aplikace jsou formuláře používané pro vytváření a editaci datových objektů. V případě, že se jedná o jednoduchý formulář, je vytvořen pomocí jediné Activity.[15] Složitější formuláře se skládají z více navazujících Activit. Typ formuláře (vytvoření/editace) se vybere dle předaných dat.

4.1.1. Přenos a ukládání dat

Data se mezi jednotlivými stránkami v aplikaci předávají různými způsoby. Tím nejhlavnějším je ukládání právě zpracovávaných dat do vlastní aplikační cache. Díky využití návrhového vzoru Singleton jsou již uložená data zabezpečena proti opětovné inicializaci, která by jinak mohla nastat při získávání cache pomocí dependency injection. Další využívanou variantou předávání informací v aplikaci je jejich uložení do objektu Intent. Intent je založen a spuštěn v případě, že aplikace potřebuje přejít na novou Activitu.

4.1.2. Uživatelské rozhraní

Formulář je vytvořen pomocí fragmentů.[16] Fragmenty reprezentují záhlaví stránky, formulářové prvky a tlačítka. Vlastnosti je možné nastavovat pomocí naimplementovaných metod. U některých formulářů se nacházejí i další fragmenty.

4.1.3. Validace

Validace základních formulářových prvků je prováděna pomocí externí knihovny Android Saripaar v2 [16]. Pro určení, jakým způsobem se budou jednotlivé uživatelské vstupy kontrolovat, použijeme knihovnou naimplementované anotace nebo si vytvoříme vlastní. Knihovna podporuje velké množství typů validace a vytváření vlastních chybových hlášek.

```
@NotEmpty
private EditText usernameInput;
@NotEmpty
@Password(min = 6, scheme = Password.Scheme.ALPHA_NUMERIC_MIXED_CASE)
private EditText passwordInput;
```

Obrázek 4.1: Ukázka určení validace formulářových prvků pomocí anotací.

[15] Activity – Android třída reprezentující stránku uživatele

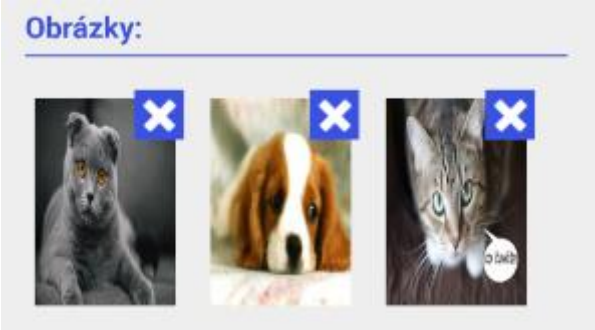
[16] Fragment – Android třída reprezentující část uživatelského rozhraní, která může být v uživatelském rozhraní na více místech.

4.2. Kamera

4.2.1. Pořizování fotografií

Jedním z požadavků aplikace je možnost přidávání fotografií a obrázků k produktům. Aplikace k tomu využívá kameru mobilního telefonu. Proto je potřeba nastavit aplikaci potřebná práva pro práci s kamerou a možnost čtení ze souborového systému zařízení.

Aplikace pro zobrazení a mazání fotek/obrázků využívá mnou naimplementovaný kontejner. Skládá se z dílčích fragmentů, které reprezentují obal pro vložený obrázek a jeho náhled. Kontejner zobrazuje vytvořené obaly obrázků do mřížkového uspořádání. V případě, že se změní orientace aplikace, je kontejner znovu vykreslen a mřížkové uspořádání zvětšeno, aby mohlo v jednom řádku zobrazit více obrázků.



Obrázek 4.2: Kontejner na obrázky.

Získané obrázky a fotografie jsou ukládány do databáze. ORMLite neposkytuje datový typ pro uložení obrázku, proto jsem se rozhodl zakódovat data obrázku pomocí Base64, které převede binární data do textového řetězce. Následný výstup kódování je možný uložit do databáze. Toto řešení by mohlo mít nevýhodu, že uložené obrázky zabírají příliš mnoho místa na disku zařízení. Alternativou by bylo ukládat do databáze informace o umístění obrázku na disku zařízení. Nevýhodou by bylo, že při případném smazání obrázku uživatelem dojde v aplikaci k jeho neobnovitelné ztrátě. Tento přístup ukládání souborů by se dal vylepšit integrací MODE_PRIVATE při vytváření a ukládání fotografií na disk. MODE_PRIVATE je konstanta, která přidává souboru přístupová práva. Definovaná práva povolují jakékoliv operace se souborem pouze aplikaci, která soubor uložila na disk. Tímto vylepšením by se vyřešila pouze půlka popsané nevýhody, protože u souborů nahraných z galerie mobilního telefonu by možnost jejich neobnovitelné ztráty při smazání přebývala.

4.2.2. Skener čárových kódů

Skener je v aplikaci využíván pro vyhledávání existujícího zboží pomocí přiděleného čárového kódu nebo pro ulehčení vytváření produktů. K implementaci funkcionality byla využita externí knihovna Barcode scanner [17]. Použití této knihovny je velmi jednoduché. Je potřeba vytvořit novou Activitu a s ní provázat instanci třídy reprezentující skener. Posledním krokem je implementace posluchače na výsledek skenování čárového kódu. Třída umožňuje využívat funkce kamery, jako je například blesk či automatické zaostřování. Dále umožňuje specifikovat typ načítaných čárových kódů.

4.3. Zobrazování dat

Náhled informací o datových položkách aplikací se většinou promítá v listech nebo seznamech. Uživatel aplikace tak má komplexní přehled o datech jednotlivých položek. V seznamu jsou vypsány

nejdůležitější informace jednotlivých prvků. Množství a typ informací záleží na navrženém řešení. Seznamy na sebe v převážném množství případů navazují detailní popis dostupný při kliknutí na vybranou položku. Mnou implementovaná aplikace využívá pro vytváření seznamů ListView [18].

4.3.1. ListView

ListView je třída v Androidu poskytující reprezentaci dat v položkovém seznamu. Jednotlivé položky jsou do seznamu vkládány pomocí tříd implementujících rozhraní Adapter. Existují již vytvořené adaptéry například pro zobrazování řetězců. V případě, že chceme v seznamu zobrazovat data vlastních objektů nebo vytvořit specifické zobrazení jednotlivých prvků, musíme provést implementaci k tomu určeného adaptéru.

V případě, že by aplikace v sobě měla uložené velké množství položek, může dojít k výrazné ztrátě výkonu. Při otevření stránky se seznamem by bylo zařízení vytíženo získáváním potřebných dat z databáze a jejich vykreslením. Tento problém se dá vyřešit přidáním stránkování. Seznam by byl rozdělen do více částí. Oproti ListView bez stránkování by aplikace zaznamenala výrazné navýšení výkonu, protože by se vždy zpracovávala pouze menší část dat. Stránkování se dá vytvořit dvěma způsoby. Prvním je uložení všech položek do kolekce a vypočítávání indexů vybraných položek pro danou stránku. Přesto se nejedná o nejlepší řešení, jelikož je aplikační výkon ušetřen pouze od vykreslování velkého množství položek. Vylepšení dodává druhý způsob, který omezuje spolu s vykreslováním položek také jejich získávání z databáze. Stačí pouze upravit dotaz, který získává data z databáze. Dotaz je potřeba rozšířit o SQL[17] vlastnosti limit a offset. Limit určuje počet položek na stránce a offset se používá pro odsazení výstupu dotazu od prvního záznamu v databázi. Offset umožnuje získání správné množiny dat vzhledem k číslu stránky.

4.3.2. RecyclerView

RecyclerView [19] je další možnost jak omezit počet získávání a vykreslování položek seznamu. Funguje na principu zobrazení co největšího množství položek, jež se vejdou do vymezené oblasti obrazovky. Položky jsou dynamicky načítány a odebírány v případě, že uživatel roluje v obsahu RecyclerView. Tento princip využívají například aplikace pro sociální sítě, kde by načítání všech příspěvků při otevření aplikace způsobilo extrémní ztrátu výkonu a výrazně zvýšilo datový tok aplikace.

Použití RecyclerView není omezeno pouze na výpis položek seznamu. Proto je základní implementace složitější než v případě ListView. Je potřeba implementovat vlastní zobrazení položky, které je povinné dědit ze třídy RecyclerView.ViewHolder. Následně jsou vytvořená zobrazení přidána do adaptéru. Na rozdíl od ListView musí mít tento adaptér specifikovaného předka, třídu RecyclerView.Adapter. RecyclerView neví, jakým stylem chce vývojář jednotlivé prvky zobrazovat, proto musí být vložen do zobrazení layout manager.[18] Mohou být využity již definované layout managery (LinearLayoutManager, GridLayoutManager a další), nebo lze provést implementaci vlastního manageru, který bude určovat pravidla, podle kterých se budou jednotlivé prvky zobrazovat.

4.3.3. Závěr

Já jsem u seznamu produktů použil ListView rozšířené o stránkování, které optimalizuje pouze vykreslování dat. Druhý přístup by byl kvůli využití hierarchie v aplikaci příliš složitý. Stránkování se zobrazí pouze, když se v listu nachází dostatečný počet záznamů.

[17] SQL – dotazovací jazyk používaný ve většině relačních databází.
[18] Layout manager – třída určená pro vykreslování prvků dovnitř zobrazení.

4.4. Filtrování dat

V aplikaci se může, jak již bylo uvedeno, nacházet velké množství záznamů. Pro vylepšení uživatelské přívětivosti je vhodné implementovat filtrování dat pro jednotlivé položky aplikace. Použití výrazně zrychlí uživateli rychlost vyhledávání konkrétních záznamů.

4.4.1. Filterable rozhraní

První možností je rozšíření adaptéru pro výpis dat o Android rozhraní Filterable. Vývojář je tak povinen naimplementovat metodu, která vrací objekt reprezentující samotný filtr. Vytvořený objekt je potomek třídy Filter a musí obsahovat metody vykonávající získávání požadovaných dat dle poskytnutých kritérií. Výsledek je po vykonání akce předán adaptéru, který změní zobrazený výstup. Tento přístup má zásadní nevýhodu, že základní integrace rozhraní umožňuje přidání pouze jednoho filtru. Dá se to vyřešit přidáním dodatečných informací o vybraných uživatelských kritériích výběru dat přímo do adaptéru a na základě nich vracet odpovídající typ filtru.

4.4.2. Vlastní rozhraní

Kvůli složitosti předávání vyhledávacích kritérií samotným filtrům při integraci rozhraní Filterable jsem se rozhodl pro implementaci vlastního interface pro tvorbu filtrů. Rozhraní je integrováno třídami filtrů, které jsou vázaný pouze na konkrétní datovou entitu aplikace. Po vývojáři je vyžadována implementace metod provádějící získávání dat z databáze na základě poskytnutých parametrů. Při vytváření rozhraní jsem chtěl umožnit vyhledávání dat pro více uživatelských kritérií najednou, což by omezilo množství přístupů k databázi. Po zjištění náročnosti implementace této možnosti jsem se rozhodl pro filtrování pomocí jediného vstupu. V případě, že uživatel potřebuje vyhledávat položky pomocí více parametrů, jsou výsledky hledání následně sjednoceny a předány adaptéru.

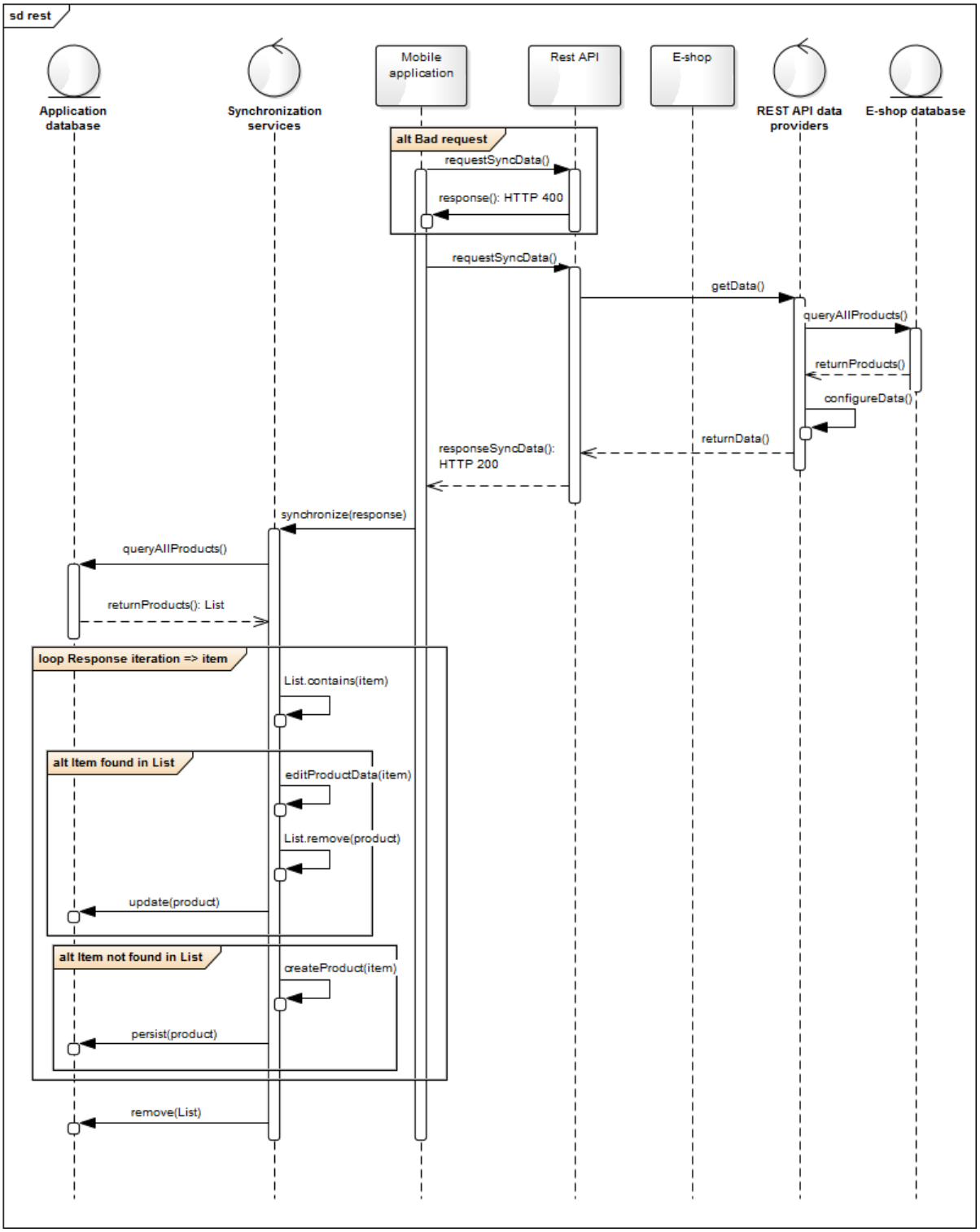
```
public interface FilterInterface <T, ID> {
    public ArrayList<T> equalsFilter(DaoInterface<T, ID> dao, String columnName, Object value);
    public ArrayList<T> notEqualsFilter(DaoInterface<T, ID> dao, String columnName, Object value);
}
```

Obrázek 4.3: Vytvořený interface pro filtrování dat.

4.5. REST API

Slouží k synchronizaci dat mezi mobilní aplikací a webovým e-shopem. Synchronizace probíhá pomocí principu znázorněného na obrázku 4.4. Sekvenční diagram popisuje pouze případ, kdy o synchronizaci zažádá mobilní aplikace. Opačný případ by byl až na drobné změny totožný.

Aplikace pošle REST rozhraní HTTP požadavek, že potřebuje synchronizovat svá data. API přepošle požadavek třídám zajišťující získávání dat pro API. Kvůli správnému návrhu rozhraní se nejedná o stejné třídy používané e-shopem pro manipulaci databázových dat. Mimo získávání dat pro rozhraní provádí také jejich úpravu. Jsou odstraněny informace, které nejsou evidovány v databázi mobilní aplikace a následně převedeny do formátu JSON. API odešle odpověď spolu s připojenými daty. Mobilní aplikace předá získaná data třídám zajišťující potřebné úpravy záznamů v databázi. Případ kdy aplikace předává své data k synchronizaci e-shopu API je simulován pomocí POST akce.



Obrázek 4.4: Sekvenční diagram postupu synchronizace dat aplikace.

4.5.1. Sandbox

Nakonec bylo rozhodnuto, že se nebude provádět synchronizace dat přímo s e-shopem. Místo toho je synchronizace simulována. K simulaci využívám službu Sandbox [20] umožňující definici REST rozhraní. Služba podporuje vytvoření rozhraní importováním kódů vybraných frameworků nebo pomocí uživatelského rozhraní a JavaScriptu. Po definici end-pointů rozhraní a jejich odpovědí je API dostupné na Internetu na přidělené adrese.

5. Programové testování

Testování aplikace je důležitá část vývoje softwaru. Slouží pro odhalení nedostatků v podobě chyb, špatného návrhu a použitelnosti. Existuje mnoho způsobů, jak testovat aplikace. V následující kapitole budou popsány metody použité pro otestování implementované aplikace.

5.1. Unit testy

Unit testy jsou naprogramované automatické testy, které kontrolují funkčnost dílčích částí aplikace. Z hlediska objektově orientovaného programování jsou tyto části obvykle reprezentovány třídami nebo konkrétními metodami. Unit testy je vhodné programovat v průběhu vytváření aplikace. Eliminují se tím chyby v základních částech aplikace, které by jinak způsobovaly nefunkčnost navazujících, komplexních implementací aplikace. Navíc je odhalování problémů v základní částech jednodušší než jejich objevování ve složitějších částech aplikace.

Na tomto druhu testů je založen i postup vývoje software. Test-driven development je metoda, kdy jsou nejdříve napsány všechny automatické testy. Až následně se vzhledem k definici těchto testů implementuje logika aplikace. Logika je upravována, dokud nedojde k bezchybnému průchodu všech testů. Výhodou tohoto principu návrhu software je existence všech automatických testů již při vytváření aplikace. To ulehčuje naleznutí chyb v případě změn ve zdrojovém kódu. Musí být zaručena bezchybnost napsaných testů. V případě, že by testy obsahovaly chyby, to může vést k chybné nebo neúspěšné implementaci.

Testování může být prováděno vzhledem k instanci aplikace spuštěné na emulátoru mobilního telefonu. Tento způsob však přináší zdlouhavý proces testování, jelikož musí dojít k načtení emulátoru, kompilaci, přeložení aplikace atd. Z tohoto důvodu se pro testování Android aplikací využívají testovací frameworky, které fungují na principu oddělení automatických testů od Android systému nebo předávání, tzv. mockování závislostí potřebných k dokončení testu.

5.1.1. Robolectric

Robolectric [21] je framework přepisující nativní Android knihovny a závislosti tak, že mohou být spuštěny na JVM. Upravené a zkompilované knihovny jsou předány jednotkovým testům a místo pomalého emulátoru jsou testy spouštěny přímo na JVM pracovní stanice, což výrazně zvyšuje efektivitu testování. Framework poskytuje funkcionality pro získávání Android zobrazení, zdrojů a mnoho dalších potřebných závislostí.

5.1.2. Mockito

Framework Mockito [22] je založený na předávání závislostí testům. Může se jednat o závislosti podporující správné fungování funkcionalit, služeb a modulů aplikace nebo závislosti určené k otestování. Při testování Android aplikací jsou ve většině případů předávány závislosti reprezentující knihovny systému Android. Odstraní se tak nutnost využití emulátoru pro testování aplikace. Mockito funguje na principu generování a vkládání kódu (mocku) používané závislosti do testu přímo v průběhu jeho vykonávání.

5.1.3. Závěr

Pro samotné vytváření unit testů jsem zvolil testovací třídu JUnit. Jedná se o nejpopulárnější a nejrozšířenější knihovnu pro vytváření jednotkových testů. K předávání závislostí jednotkovým testům jsem se rozhodl využít framework Mockito. Jedná se o jeden z nejlepších mockovacích frameworků pro

jazyk Java, hlavně díky možnosti vytváření čistých a dobře čitelných testů, zobrazení přehledných výsledků a schopnosti frameworku předávat libovolnou závislost implementovanou v jazyce Java.

5.2. Integrační testy

Tento typ testů se zaměřuje na testování komunikace a správného využívání jednotlivých částí aplikace, které byly testovány pomocí jednotkových testů. Může se jednat o komunikaci mezi jednotlivými částmi aplikace nebo mezi částí a operačním systémem. Proto je tento typ testů spouštěn proti emulovanému nebo standardnímu Android zařízení. Při vývoji testů je potřeba dbát zvýšené obezřetnosti při manipulaci s daty a nutnosti implementace mechanismu pro navrácení dat v databázi do původní podoby po skončení testu. Kdyby se vývojář integračních testů řídil tímto postupem, je vysoká pravděpodobnost existence chyby testu, která by vyústila zanesením databáze nesmyslnými daty nebo poškozením integrity dat stávajících. Tomuto problému se dá vyhnout využitím technologií, které eliminují používání aplikační databáze v integračních testech.

5.2.1. H2 databáze

Jednou z metod zamezení poškození dat aplikace testy je využití H2 databáze [23]. Tato databázová technologie funguje na principu in-memory databází. To znamená, že struktura databáze a její data jsou ukládány do paměti. Paměťová náročnost databáze je dobře optimalizována, a nebude tedy výrazně ovlivňovat rychlost prováděných testů.

H2 databáze je open source řešení naprogramované v jazyce Java. Mezi klíčové vlastnosti patří vysoká rychlost databázového enginu, velmi dobré zabezpečení a podpora jazyka SQL. Z hlediska SQL podpory umožňují H2 databáze většinu základních a pokročilých funkcionalit dostupných v běžně používaných databázích (PostgreSQL, MySQL). Jedná se například o podporu více schémat, triggery, sekvence pro automatickou inkrementaci sloupců, podporu velkého množství datových typů včetně objektů a polí.

Technologie podporuje tři způsoby využití. Nejzákladnější mód se nazývá Embedded a databáze je spuštěna v rámci aplikace. Aplikace pomocí JDBC[19] otevře připojení k databázi. Databáze musí být vytvořena ve stejném JVM jako spuštěná aplikace. Tento způsob připojení je charakteristický svou rychlostí. Dalším typem použití H2 databází je Server mode. Zde se databáze nachází na serveru, což umožňuje více aplikacím se připojovat ke společné in-memory databázi. Tento typ připojení je pomalejší než základní Embedded režim, jelikož se pro komunikaci používá TCP/IP protokol, který přidává k vytvoření spojení a výměně dat režii. Posledním typem je Mixed mode, který je kombinací předchozích.

Jelikož je H2 databáze potřeba pouze pro integrační testy, je dostačující využití Embedded módu. Je tedy potřeba definovat výše zmiňované JDBC spojení, které jednoznačně určuje používanou databázi. H2 databáze podporuje object-relation mapping. Pro generování tabulek je tedy možné použít entity, které mají databázové sloupce identifikované pomocí JPA nebo ORMLite anotací. ORMLite využívá tříd DAO, které poskytují základní operace pro manipulaci dat definované databáze. V aplikaci jsou implementovány komponenty, které ke správné funkci tyto třídy potřebují. Při testování těchto komponent se tedy musí vytvořit nové DAO třídy, které budou pracovat se správnou databází. DAO třídy se v komponentách musí nahradit. Tím dosáhneme, že testy nevyužívají aplikační databázi.

[19] JDBC – rozhraní v jazyce Java určené pro vytváření připojení k relačním databázím.

```
private JdbcConnectionSource createDatabaseConnection(String appNamespace, String dbName) {
    try {
        return new JdbcConnectionSource(
                "jdbc:h2:/data/data/" + appNamespace + "/databases/" + dbName);
    } catch (SQLException e) {
        throw new RuntimeException("Connection source to database cannot be created. ", e);
    }
}
```

Obrázek 5.1: Vytvoření spojení H2 databáze.

```
private void createDatabaseTables(ConnectionSource connectionSource) {
    try {
        TableUtils.createTable(connectionSource, User.class);
        TableUtils.createTable(connectionSource, Address.class);
        TableUtils.createTable(connectionSource, StoragePosition.class);
        TableUtils.createTable(connectionSource, ProductCategoryRelationship.class);
        TableUtils.createTable(connectionSource, Parameter.class);
        TableUtils.createTable(connectionSource, ParameterType.class);
        TableUtils.createTable(connectionSource, Barcode.class);
        TableUtils.createTable(connectionSource, Manufacturer.class);
        TableUtils.createTable(connectionSource, Notification.class);
        TableUtils.createTable(connectionSource, Picture.class);
        TableUtils.createTable(connectionSource, Product.class);
        TableUtils.createTable(connectionSource, Category.class);
    } catch (SQLException e) {
        throw new RuntimeException("Some database tables couldn't be created. ", e);
    }
}
```

Obrázek 5.2: Metoda pro generování databázové struktury.

6. Testování uživatelského rozhraní

6.1. Testování bez uživatelů

Jedná se o metody testující použitelnost [24] aplikace a jejího uživatelského rozhraní. Hlavní výhodou je nízká náročnost na čas a personál potřebný k realizaci. Některé typy testů je vhodné vypracovat po dokončení analýzy. Jsou tak nalezeny potencionální chyby na prototypu uživatelského rozhraní. Chyby se opraví před jejich implementací, což ušetří spoustu času a práce v průběhu vývoje.

6.1.1. Kognitivní průchod

Kognitivní průchod je metoda testování uživatelského rozhraní. Základem jsou testovací scénáře popisující akce uživatele vedoucí k dokončení určitého cíle. Tester se při každém kroku scénáře ptá na následující otázky.

a) Bude uživatel vědět jak správně zareagovat?
b) Bude uživatel vědět jak to udělat?
c) Obdrží uživatel zpětnou vazbu?

Jako příklad je uveden test kognitivním průchodem pro přidání nového produktu. Znak X u výsledků otázky c) znamená, že se uživatel nachází na stejné stránce, a proto mu nebyla poskytnuta zpětná vazba.

Testovací scénář

1. Otevření stránky Zboží a kategorie z hlavního menu nebo úvodní stránky aplikace.
2. Kliknutí na tlačítko pro vytvoření nového záznamu.
3. Vybrání možnosti Přidat produkt.
4. Vyplnění formuláře pro získání základních informací o produktu.
5. Spuštění akce pro přidání obrázku k produktu.
6. Vybrání preferované možnosti pořízení obrázku.
7. Vyfocení nebo nahrání obrázku.
8. Naskenování čárového kódu identifikujícího zboží.
9. Odeslání formuláře.
10. Potvrzení dalšího nastavování produktu.
11. Vybrání příslušné kategorie a výrobce produktu.
12. Vyplnění formuláře definujícího dodatečné informace o produktu.
13. Odeslání formuláře.
14. Vyplnění formuláře pro konfiguraci parametrů vybrané kategorie.
15. Dokončení vytváření produktu.

Test

	Kroky testovacího scénáře														
	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.	13.	14.	15.
a)	✓	✓	✓	✓	1)	✓	✓	2)	✓	✓	✓	✓	✓	4)	✓
b)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	3)	✓	✓	5)	✓
c)	✓	✓	✓	X	✓	✓	✓	✓	✓	✓	✓	X	✓	✓	✓

Tabulka 6.1: Výsledky kognitivního průchodu.

Nalezené chyby

1) Chybějící popis tlačítka (Priorita: nízká)

Tlačítko pro nahrávání obrázku obsahuje pouze ikonku fotoaparátu. Uživatel si tak nemusí být jistý exaktním účelem tlačítka. Ikonka dodává uživateli celkem dobrou reprezentaci nastávající akce po kliknutí na tlačítko, proto je nálezu přiřazena nízká priorita. Řešením tohoto problému by mohlo být přidání názorného popisku do stávající/nové ikonky nebo přidání nápovědy. Nápovědu je možné zobrazit a skrýt na základě uživatelské interakce.

2) Nesrozumitelná ikonka tlačítka (Priorita: střední)

Skenování čárového kódu se spouští pomocí tlačítka, které obsahuje ikonku znázorňující znak plus. To může být pro prvotní uživatele velmi nesrozumitelné a zavádějící. Odstranění chyby lze nejlépe provést změnou ikonky tlačítka. V případě, že nová ikonka také zcela nedefinuje akci skenování čárového kódu, je vhodné přidat popis nebo nápovědu pro uživatele.

3) Nedostačující reprezentace dat při výběru kategorie (Priorita: vysoká)

Formulářový prvek zajišťující výběr kategorie může být při velkém množství možností nepřehledný. Všechny kategorie jsou vypsány stejným způsobem. Uživatel tak nemá informaci o tom, zda si vybral hlavní kategorii nebo pouze některou z podkategorií. Dalo by se to odhadnout z kontextu a porovnání jmen kategorií, to však nemusí být přesné. Možnou eliminací tohoto nedostatku je integrace hierarchie kategorií do zobrazení možností výběru. Hlavní kategorie bude např. vypsaná v seznamu tučně a bude následována jejími podkategoriemi, které by byly odsazeny, stylovány nebo číslovány na základě úrovně v hierarchii.

4) Neexistující popis parametru (Priorita: střední)

Obsahuje-li některá z kategorií parametry, jejichž jména přesně neidentifikují účel parametru, bude mít uživatel potíže při vytváření nového záznamu o zboží. Priorita opravy je určena pouze jako střední. To je dáno předpokladem, že pracovníci skladů budou aplikaci používat zodpovědně a nebudou vytvářet nesmyslné jména parametrů. Návrh řešení problému je přidání nápovědy popisující parametr.

5) Chybějící reprezentace datového typu parametru (Priorita: vysoká)

Tento nález souvisí s chybou označenou pomocí 4). Nejedná se však o porozumění parametru, ale o množinu hodnot, které mohou být k parametru přiřazeny. Aplikace umí na základě datového typu parametru poznat, zda hodnota má být číslo, a vzhledem k tomu nastaví na formulářový prvek odpovídající klávesnici. Aplikace nerozlišuje mezi pravdivostní a textovou hodnotu parametru. Vstup pro hodnotu je textový, což při chybné interpretaci pravdivostní hodnoty může způsobit nefunkčnost aplikace. Odstranění chyby může být provedeno vytvořením nového výběrového formulářového prvku, který je vykreslen pro parametry s pravdivostním datovým typem.

6.1.2. Heuristická evaluace

Metoda Heuristické evaluace kontroluje, zda aplikace správně dodržuje předem definované pravidla (heuristiky). Pro tuto metodu bylo vytvořeno již mnoho typů pravidel, ze kterých si tester může vybrat, případně si některé pravidla upravit či nadefinovat vlastní. Pro testování implementované aplikace jsem vybral heuristiky definované Jakobem Nielsenem [25].

a) Visibility of system status – systém informuje v dostatečném čase uživatele o tom, co se v systému děje.
b) Match between system and real world – systém s uživatelem komunikuje stylem, který je pro danou skupinu uživatelů nejsrozumitelnější. Informace musí být logicky zobrazovány jako v reálném světě.
c) User control and freedom – systém musí poskytovat uživateli dobře viditelný návrat z nechtěných situací.
d) Consistency and standards – systém musí dodržovat standardy a předpisy definované platformou, ve které se nachází.
e) Error prevention – systém by měl být navržen tak, aby uživatel nedělal zbytečné chyby. Příkladem mohou být potvrzovací okna pro uživatelské akce, správně popsané kroky atd.
f) Recognition rather than recall – uživatel by si neměl pamatovat informace mezi jednotlivými kroky. Systém musí být schopen tyto informace uživateli vypsat. Vše by mělo být v systému jasně popsané.
g) Flexibility and efficiency of use – systém by měl obsahovat možnosti používání pro více zkušené uživatele.
h) Aesthetic and minimalist design – dialogy by neměly zobrazovat nerelevantní nebo nepotřebné informace. Čím více systém zobrazuje informací, tím je nižší šance, že uživatel nalezne potřebné informace.
i) Help users recognize, diagnose and recover from errors – chybové hlášky musí být napsány v pro uživatele pochopitelném jazyce, musí jednoznačně indikovat problém a navrhnout řešení.
j) Help and documentation – systém by měl mít vytvořenou uživatelskou příručku popisující práci se systémem. Dokumentace by měla být lehce k nalezení a neměla by být příliš dlouhá.

Nalezené chyby

1) Absence nápovědy u formulářových prvků (Priorita: střední)

Některé formulářové vstupy mohou být označovány ne zcela srozumitelným popiskem. Uživatel tak může vyplnit chybná či nerelevantní data, která budou v databázi dělat nepořádek. Dále mohou mít formulářové prvky přiřazená pravidla, pomocí kterých je prováděna kontrola vyplněných hodnot při odeslání formuláře. V případě, že je validace neúspěšná, aplikace vypíše chybové hlášky. To je v pořádku, ale lepší by bylo chybné validaci předejít. Popsané problémy mohou být vyřešeny přidáním nápovědy deklarací správného typu a tvaru přidávaných hodnot. Nápovědu je možné zobrazit a skrýt na základě uživatelské interakce.

2) Aplikace neupozorňuje uživatele na docházející zboží (Priorita: nízká)

V případě, že počet produktů klesne pod minimální povolené množství, uživatel se to může dozvědět v aplikaci pouze na stránce Upozornění. Včasné zjištění nedostatku produktů a jejich doobjednání je tedy zcela závislé na pravidelnosti navštěvování přehledu docházejícího zboží uživatelem. Mohou tak nastat situace, kdy na skladě bude některé zboží chybět delší dobu a firma přijde o zisky z prodeje. K odstranění této nechtěné situace se dá využít generování upozornění pro systém Android. Upozornění může být vytvářeno jako kompletní přehled v předem stanoveném čase nebo pouze jako jediný záznam přímo v okamžiku snížení produktu pod povolené množství. Výhodou tohoto typu upozornění je, že jsou produkována i v případech, kdy uživatel nepoužívá aplikaci ani mobilní telefon.

3) Chybějící identifikace chybových hlášek (Priorita: střední)

Aplikace pro sdělování zpětné vazby o prováděných akcích používá Android třídu Toast. Třída slouží k vypsání libovolné zprávy, která bude zobrazena pro určený časový interval ve spodní části obrazovky. Aplikace nerozlišuje mezi chybovou a normální zprávou a obě jsou vykresleny stejným způsobem. Možností na vyřešení problému je implementace vlastního zobrazení, které bude například disponovat jinou barvou.

4) Absence detailních informací v chybových hláškách (Priorita: střední)

V případě, že v systému nastane chyba, je o ní uživatel informován pomocí již zmiňované třídy Toast. Jedná se pouze o základní zprávu, která neobsahuje důvod nastání chyby ani nápovědu, jakým způsobem postupovat, aby vybraná funkcionalita proběhla úspěšně. Je tedy potřeba chybovou hlášku rozšířit o spoustu textu. Příčina chyby aplikace se dá zjistit vyhazováním a ověřováním různých typů výjimek. Z příčiny navrhneme uživateli možný postup pokračování. Chybová hláška je nyní dlouhý text a ty je nevhodné vypisovat pomocí Toast. Může totiž nastat situace, že text zmizí dříve, než si ho uživatel stihne přečíst. Řešením může být přidání zobrazení pro vykreslování a výpis hlášek na všechny stránky aplikace. Kdyby potřebovala aplikace něco sdělit uživateli, tak by se zobrazení vykreslilo spolu s kompletní zprávou a zmizelo by až při obnovení stránky nebo při přesunu na jinou.

5) Nefunkční nastavení základní hodnoty pro výběr kategorie (Priorita: vysoká)

Uživatel se nachází na stránce Zboží a kategorie, kde si otevře detail některé kategorie. V této kategorii se rozhodne vytvořit nový produkt. Vyplní základní informace a rozhodne se, že chce pokračovat v nastavování produktu. Aplikace ho tedy přenese na nový formulář, kde se nachází mimo jiné i prvek pro výběr kategorie. Tento prvek nemá nastavenou jako vybranou hodnotu kategorii, kterou uživatel vybral průchodem v seznamu. Musí si tak pamatovat, kterou kategorii vybral, a znovu ji nastavit. Řešením by bylo upravení pořadí dat v adaptéru, který se používá pro vykreslení možností do formulářového prvku. Vybraná kategorie ze seznamu by se musela najít v kolekci adaptéru a prohodila si místo s prvkem na prvním místě.

6.1.3. Závěr

Pomocí těchto dvou testování metod byly nalezeny některé nedostatky aplikace. Nejsou to chyby, které znemožňují správné fungování aplikace, ale jejich oprava by zlepšila uživatelskou přívětivost. V dokumentu nebyly sepsány všechny nedostatky aplikace, ale pouze ty nejhlavnější. Některé z naleznutých problémů byly opraveny na základě jejich navrhnutého řešení.

6.2. Testování s uživateli

Toto testování bylo realizováno až po kompletní implementaci aplikace. Testování s uživateli je náročnější na čas a potřebné zdroje. Probíhá většinou v laboratoři se speciálním vybavením a personálem dohlížejícím na průběh testu. Hlavní podmínkou pro start testů je sehnání účastníků. K tomu je potřeba vytvoření dotazníku, tzv. screeneru, který obsahuje otázky zjišťující informace o uživatelích a jejich zkušenostech s danou problematikou. Dotazník je poslán co nejvíce lidem cílové skupiny aplikace. Následně je screener vyhodnocen a na test jsou pozváni jeho úspěšní řešitelé.

6.2.1. Nalezené chyby

Kvůli ušetření využitého místa dokumentu a eliminaci duplicitních popisů evidovaných chyb jsem se rozhodl, že v této kapitole budou vypsány pouze problémy, které nejsou uvedené v předchozí kapitole. Část chyb nalezených uživateli způsobovaly nefunkčnost aplikace. Funkční aplikace je důležitější. Díky tomu jsem se snažil opravit co nejvíce nalezených chyb způsobujících nefunkčnost aplikace a u popisu chyby jsem mohl místo navrhovaného řešení popsat reálně provedené opravy.

1) Nefunkční přihlášení pomocí nově vytvořeného uživatelského účtu (Priorita: vysoká)

Aplikace neumožňuje přihlášení uživateli, který si vytvořil nový účet. Po odeslání formuláře se objeví chybová hláška, že je heslo neplatné. Přihlášení pomocí administrátorského účtu funguje. Problém byl v generátoru pro vytváření hesel. Na přihlašovacím formuláři je na heslo nastavená validace, že musí obsahovat alespoň šest znaků, jedno číslo a jedno velké písmeno. Generátor vytvářel heslo pomocí přístupů na náhodné pozice řetězce, který sjednotil všechny znaky abecedy a čísla. Náhoda zapříčinila, že bylo vygenerováno heslo nesplňující kritéria a přihlašovací formulář vždy skončil validační chybou. Problém byl vyřešen náhodným výběrem prvního znaku hesla pouze z množiny velkých písmen a na poslední místo pouze z množiny čísel.

2) Posun tlačítka na přihlašovací stránce (Priorita: vysoká)

Při vstupu uživatele na přihlašovací stránku je vše v pořádku. Problém nastane, až když uživatel začne vyplňovat uživatelské jméno. Tlačítko pro přihlášení se přesune doprostřed stránky a překrývá vstup pro heslo. Problém byl ve špatně navrženém zobrazení stránky. Používal jsem k vykreslení tlačítka na spodní část stránky třídu, která umožňuje zarovnat prvek na střed, vertikálně nebo horizontálně. V případě, že se otevřela klávesnice, došlo ke zmenšení obrazovky a tím se přesunulo i zarovnané tlačítko. Vyřešil jsem to odstraněním třídy na zarovnávání a odsadil tlačítko od formuláře.

3) V detailu uživatele se nezobrazují data popisující adresu uživatele (Priorita: vysoká)

Při zobrazení detailu uživatele chybí informace o jeho adrese. Chyba nastává kvůli ne zcela funkční implementaci Object-relation mappingu pomocí JPA. Vytváření nových databázových záznamů funguje a umožňuje tabulky mezi sebou propojovat pomocí nastavení cizího klíče. I přes vytvořenou vazbu mezi uživatelem a adresou není při dotazu na získání uživatele z databáze automaticky s ním vytažena i adresa. Když byla databázová struktura definována pomocí ORMLite anotací, tak vše fungovalo v pořádku. Možností vyřešení chyby je vytvoření nového databázového dotazu pro získání adresy uživatele nebo opravení databázové struktury.

6.2.2. Závěr

Uživatelé při testování nalezli mnoho problémů s použitelností uživatelského rozhraní, které byly naleznuty už v testování bez uživatelů. Kvůli nedostatečnému času byla aplikace uživateli testována ve fázi, kdy některé funkce nebyly plně dodělány.

7. Závěr

Vypracování práce zlepšilo mé dovednosti programování nativních Android aplikací. Získání těchto zkušeností je v dnešní době velmi prospěšné, jelikož většina lidí vlastní smartphone. Díky tomu se trh s mobilními aplikacemi stal velmi lukrativním. Dále jsem se toho hodně dozvěděl o technologiích používaných v dnešní době. Ať už pomocí jejich využití při implementaci některých problematik nebo jejich studiem a popsáním v dokumentu. Získal jsem zkušenosti s používáním frameworků a knihoven třetích stran. Dále jsem si připomněl již nabyté vědomosti ohledně testování aplikací.

Cílem práce bylo vytvoření aplikace pro správu skladu s funkcionalitami pro evidenci zboží, přidávání fotografií a synchronizace dat. První částí byla analýza, začal jsem zkoumáním již existujících řešení aplikace pro správu skladů. To mi přineslo inspiraci na vytvoření funkcionalit, které budou uživateli ulehčovat práci s aplikací nebo poskytovat rozšířené možnosti. Mezi některé nápady patří načítání produktu pomocí čárového kódu, hierarchie kategorií či definice parametrů specifických pro danou kategorii. Následovalo vytvoření diagramu reprezentujícího strukturu dat, vymyšlení požadavků, vytvoření scénářů použití a navrhnutí uživatelského rozhraní. Část analýzy se mi dle mého názoru povedla.

Co se týče implementace, tam jsem narazil na pár problémů. Tím největším bylo špatné odhadnutí doby trvání vývoje aplikace. Z tohoto důvodu jsem musel vynechat nebo zjednodušit některé funkcionality navržené v analýze. Co se týče zadání práce, tak jsou implementovány veškeré požadavky. Pouze synchronizace dat s e-shopem byla nasimulována pomocí služby na tvorbu REST rozhraní. Kompletní synchronizace s e-shopem nebyla realizována z důvodu časového zaneprázdnění jak z mé strany, tak i ze strany studenta, který vytvářel e-shop.

Použití některých technologií mi přineslo nepříjemnosti z hlediska fungování aplikace. Tím největším problémem byl přechod z ORMLite anotací na JPA pro tvorbu databázové struktury. Pro JPA nefungovalo při získávání objektu z databáze automatické načítání objektu, který je spojen s dotazovaným pomocí vazby realizované pomocí cizího klíče. Musel jsem tak většinou vytvářet další dotazy, abych získal i tyto data. Z tohoto důvodu jsem se rozhodl navrátit zpět na vytváření struktury pomocí ORMLite anotací. Při implementaci jsem také udělal některá špatná rozhodnutí. Jedním z nich je nepoužití Firebase databáze, která by mi zaručila synchronizaci dat mezi aplikacemi. Dalším bylo vytvoření vlastních filtrů, po snaze implementace slučování filtrů a spousty dalších věcí jsem zjistil, že to je příliš obtížné. Došlo tedy ke zjednodušení a nakonec fungují stejně jako Androidem poskytované rozhraní, takže jsem pro implementaci mohl použít to.

Do aplikace budou v budoucnu přidány analýzou navržené funkcionality, které nestihly být včas do aplikace integrovány. Další možnou funkcionalitou je vytváření záloh dat a možnost jejich ukládání na cloudové úložiště. Aplikace by mohla být rozšířena o nový modul. V případě, že si uživatel vybere verzi aplikace s tímto modulem, byla by mu umožněna funkcionalita i z administrativního hlediska. Mohl by tak spravovat faktury, dokumenty a další věci spojené s vedením skladu nebo obchodu.

8. Reference

[1] LACKO, L.: Vývoj aplikací pro Android. Brno: Computer Press, první vydání, 2015, ISBN 978-80-251-4347-6.

[2] About SQLite. *Sqlite* [online]. SQLite Consortium [cit. 2018-05-12]. Dostupné z: https://sqlite.org/about.html

[3] WATSON, Gray.: OrmLite Package [online]. Version 5.0, July 2016. [cit. 2018-05-12]. Dostupné z: http://ormlite.com/docs/ormlite.pdf

[4] Balsamiq Products. *Balsamiq* [online]. Balsamiq Studios [cit. 2018-05-12]. Dostupné z: https://balsamiq.com/products/

[5] Kotlin Programming Language. *Kotlin* [online]. JetBrains [cit. 2018-05-12]. Dostupné z: https://kotlinlang.org/

[6] <uses-sdk> | Android Developers. *Android developers* [online]. Google Developers [cit. 2018-05-12]. Dostupné z: https://developer.android.com/guide/topics/manifest/uses-sdk-element

[7] Support Library | Android Developers. *Android developers* [online]. Google Developers [cit. 2018-05-12]. Dostupné z: https://developer.android.com/topic/libraries/support-library/

[8] Introduction to the Java Persistence API | The Java EE 6 Tutorial. [online]. Oracle. 2013 [cit. 2018-05-12]. Dostupné z: https://developer.android.com/guide/topics/manifest/uses-sdk-element

[9] Firebase Realtime Database|Store and sync data in real time. [online]. Google Developers [cit. 2018-05-12]. Dostupné z: https://firebase.google.com/products/realtime-database/

[10] Dependency Injection. *Best Practice Software Engineering* [online]. Information & Software Engineering Group, 2013 [cit. 2018-05-12]. Dostupné z: http://best-practice-software-engineering.ifs.tuwien.ac.at/patterns/dependency_injection.html

[11] Roboguice | Wiki. *Github* [online]. 2014 [cit. 2018-05-20]. Dostupné z: https://github.com/roboguice/roboguice/wiki

[12] WHARTON, Jake. Butter Knife. *Github* [online]. [cit. 2018-05-20]. Dostupné z: http://jakewharton.github.io/butterknife/

[13] Dagger | User's Guide. *Google github* [online]. Google Developers [cit. 2018-05-20]. Dostupné z: https://google.github.io/dagger/users-guide

[14] What is a REST API?. *MuleSoft* [online]. MuleSoft [cit. 2018-05-20]. Dostupné z: https://www.mulesoft.com/resources/api/what-is-rest-api-design

[15] *GraphQL* [online]. Facebook [cit. 2018-05-20]. Dostupné z: http://graphql.org/

[16] Android Saripaar v2. *Github* [online]. Mobs & Geeks [cit. 2018-05-20]. Dostupné z: https://github.com/ragunathjawahar/android-saripaar

[17] Barcodescanner. *Github* [online]. [cit. 2018-05-20]. Dostupné z: https://github.com/dm77/barcodescanner

[18] List View| Android Developers. *Android developers* [online]. Google Developers [cit. 2018-05-20]. Dostupné z: https://developer.android.com/guide/topics/ui/layout/listview

[19] Create a List with RecyclerView | Android Developers. *Android developers* [online]. Google Developers [cit. 2018-05-20]. Dostupné z: https://developer.android.com/guide/topics/ui/layout/recyclerview

[21] *Sandbox* [online]. Sandbox [cit. 2018-05-23]. Dostupné z: https://getsandbox.com/

[21] *Robolectric* [online]. Robolectric [cit. 2018-05-20]. Dostupné z: http://robolectric.org/

[22] *Mockito* [online]. [cit. 2018-05-20]. Dostupné z: http://site.mockito.org/

[23] H2 Features. *H2 Database Engine* [online]. [cit. 2018-05-20]. Dostupné z: http://www.h2database.com/html/features.html

[24] HOLZINGER, A.: Usability engineering methods for software developers. Austria: Graz Univerzity. 2005. doi>10.1145/1039539.1039541.

[25] 10 Usability Heuristics for User Interface Design. *Nielson Norman Group* [online]. Nielson Norman Group [cit. 2018-05-20]. Dostupné z: https://www.nngroup.com/articles/ten-usability-heuristics/

9. Přílohy

K bakalářské práci bylo přiloženo CD obsahující zdrojové kódy aplikace, dokument s vysvětlivkami zkratek a termínů a příručku pro uživatele. Soubory jsou v cd rozděleny následujícím způsobem.

- src: zdrojové kódy aplikace
- build: zkompilované zdrojové kódy aplikace
- api: zdrojové soubory použité na vytvoření REST API ve službě SandBox
- bakalarska_prace.docx
- bakalarska_prace.pdf
- vysvetlivky.pdf
- manual.pdf