Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Mobilní pomocník zaměstnance knihkupectví

Pro firmu Knihy Dobrovský spol. s.r.o.

2020

Michal Trojek

Vedoucí práce: Mgr. Petr Osička, Ph.D.

Studijní obor: Aplikovaná informatika, kombinovaná forma

Bibliografické údaje

Autor:	Michal Trojek
Název práce:	Mobilní pomocník zaměstnance knihkupectví (Pro firmu Knihy Dobrovský spol. s.r.o.)
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2020
Studijní obor:	Aplikovaná informatika, kombinovaná forma
Vedoucí práce:	Mgr. Petr Osička, Ph.D.
Počet stran:	43
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliographic info

Author:	Michal Trojek
Title:	Mobile application for bookstore employees
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2020
Study field:	Applied Computer Science, combined form
Supervisor:	Mgr. Petr Osička, Ph.D.
Page count:	43
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Práce je zaměřena na vývoj mobilní aplikace pro zaměstnance knihkupectví. Hlavním důvodem jejího vzniku byla snaha o zefektivnění práce zaměstnanců a urychlení získávání informací o produktech, prodeji a stavech skladů. Aplikace je psaná v programovacím jazyku Java a je vytvořena pro operační systém Android.

Synopsis

The thesis is focused on the development of a mobile application for bookstore employees. The main reason for its creation was an effort to streamline the work of employees and to speed up the acquisition of information on products, sales and stock levels. The application is written in Java programming language and is designed for the Android operating system.

Klíčová slova: Android, Java, MVVM, JavaFX, Windows, SQLite

Keywords: Android, Java, MVVM, JavaFX, Windows, SQLite

Tímto bych chtěl poděkovat vedoucímu bakalářské práce panu Mgr. Petrovi Osičkovi, Ph.D. za odborné vedení při vypracovávání této práce.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Za cíl své bakalářské práce jsem si stanovil vytvoření aplikace, která usnadní práci zaměstnancům firmy Knihy Dobrovský spol. s.r.o., práce se skládá ze dvou částí.

První částí je hlavní aplikace pro mobilní telefony, která zpracovává a zobrazuje data z analýzy prodejů a informace o stavech skladů.

Druhá část je podpůrný program běžící na počítači, který slouží k ručnímu nahrávání dat z analýzy prodejů do hlavní aplikace. Tento podpůrný program se používá jen pokud uživatel nemá přístup k datům na FTP serveru.

Obě části spolu komunikují pomocí lokální bezdrátové počítačové sítě.

1.1 Příklad použití

Aplikace je užitečná především pro vedoucí pracovníky, kteří díky ní budou mít neustálí přístup k datům o prodejnosti knih a stavech skladů na všech prodejnách.

Aplikace umožňuje následující scénář:

Uživatel si do telefonu nahraje data z analýzy prodejů některé prodejny a pomocí aplikace zjistí informace (prodejnost, umístění apod.) o některé z knih.

Knihu identifikuje naskenováním čárového kódu, anebo alternativně fulltextovým vyhledáváním názvu knihy v databázi. Poté se uživatel může rozhodnout danou knihu zařadit do vratky, případně knihu doobjednat.

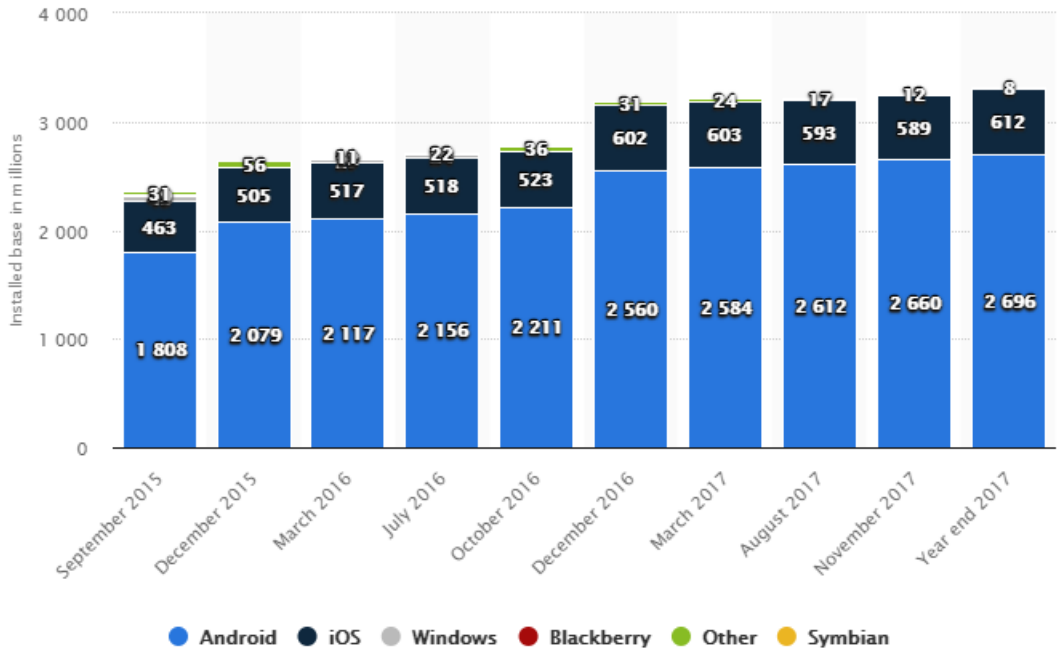
Aplikace uživateli nabídne možnost objednávku vytisknout přes wifi tiskárnu anebo vytvořit excelový soubor a ten poté odeslat emailem.

2 Návrh systému

V této části popisuji architekturu celého systému a použité technologie. Při výběru technologii jsem zohlednil požadavek zástupce firmy Knihy Dobrovský aby aplikace fungovala i na starších čtečkách čárových kódů, které běží na operačním systému *Android*.

2.1 Android

Android je operační systém založený na jádře Linuxu vytvořený firmou *Android Inc*, která byla koupena firmou *Google* v roce 2005. Obrovskou výhodou tohoto operačního systému je jeho rozšířenost. V roce 2017 měl *Android* 85% podíl na trhu chytrých telefonů a přes 2,7 bilionů uživatelů oproti svému hlavnímu konkurentovi *IOS* od společnosti *Apple* s 612 miliony uživatelů. [statista][2].



Obrázek 1: *Podíl na trhu chytrých telefonů*

Důvodů pro takovou popularitu je hned několik.

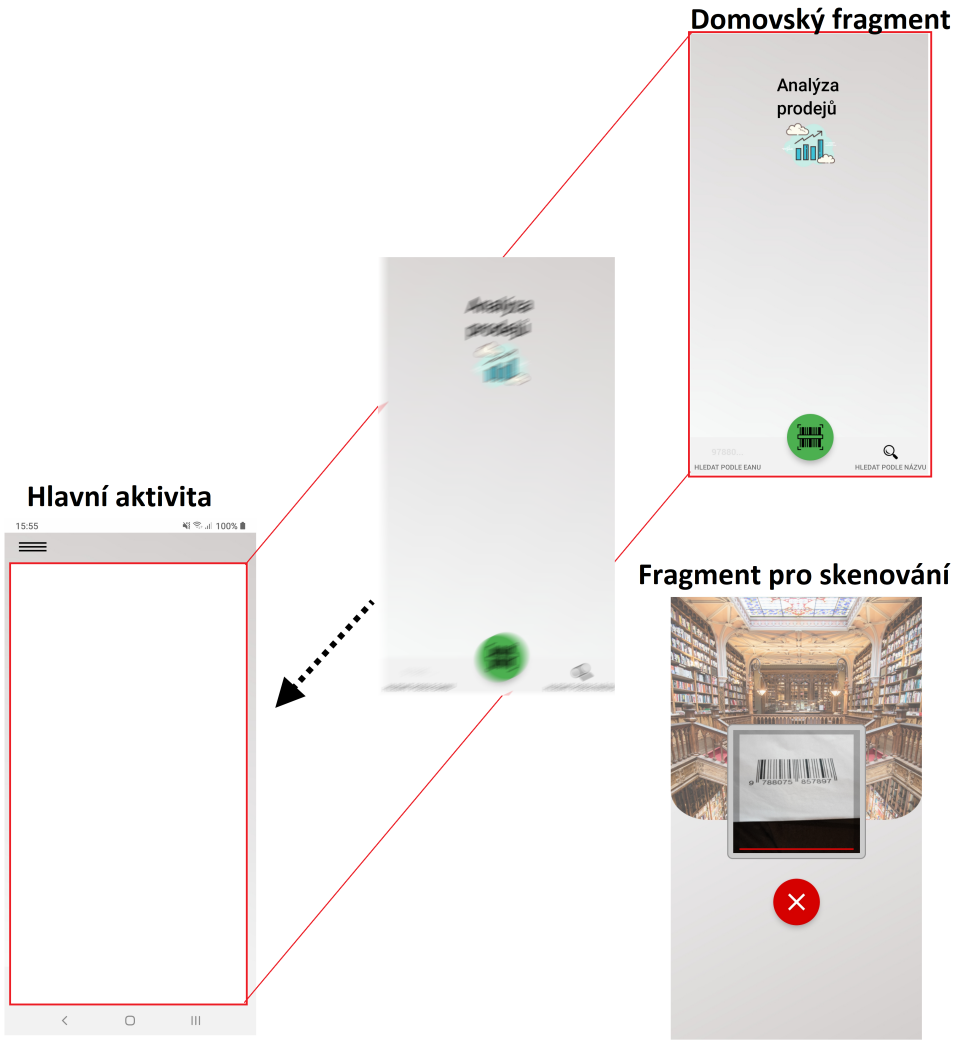
- *Android* je vlastněn společností *Google* a má neustálou podporu.
- *Android* je otevřený software, který je snadno upravitelný a proto ho používají i jiné velké společnosti.
- *Android* je dostupný na zařízeních v každé cenové kategorii.

2.2 Aktivity nebo fragmenty?

Na začátku jsem se musel rozhodnout, zda se aplikace bude skládat z několika aktivit, anebo pouze z jedné vstupní aktivity a několika fragmentů.

Aktivita reprezentuje jednu celou obrazovku aplikace a fragment je znovu použitelná část grafického rozhraní a logického kódu, kterou je možné podle potřeby vkládat do a odebírat z aktivity.

Rozhodl jsem se aplikaci vytvořit způsobem jedna vstupní aktivita a několik fragmentů, protože je to způsob doporučený *Android* týmem a příležitost vyzkoušet jak funguje navigační komponenta[1] ze sady *Android Jetpack Components*[2].



Obrázek 2: *Hlavní aktivita a fragmenty*

[1]Navigation component https://developer.android.com/guide/navigation/
[2]Android jetpack components https://developer.android.com/jetpack

2.3 Architektura Model-View-ViewModel

Tuto architekturu jsem vybral, protože je to architektura doporučena *Android* týmem pro vývoj aplikací pro operační systém *Android*.

Je to architektura rozdělující aplikaci do tří logických vrstev, kde každá vrstva má na starosti jednu úlohu a díky tomu je možné vrstvy jednodušeji upravovat a měnit.

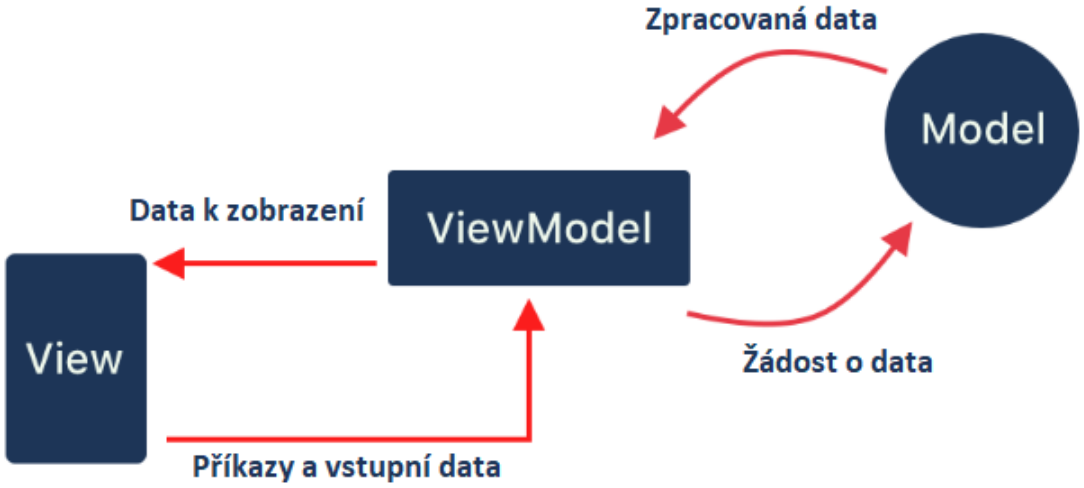
Model je vrstva, která by měla komunikovat pouze s vrstvou *ViewModel*. V této vrstvě jsou ty třídy, které mají na starosti zdrojová data a byznys logiku aplikace. Tato data jsou potom předávaná vrstvě *ViewModel*.

V mé aplikaci se jedná a třídy, které pracují s *SQLite* databází pomocí knihovny *Room*[3].

View reprezentuje uživatelské rozhraní. Má na starosti zobrazování informací uživateli, informovat vrstvu *ViewModel* o změnách a předávat jí vstupní data od uživatelů. Je zde vytvořená reference na jednu anebo víc tříd typu *ViewModel* a také jsou tu reference na speciální proměnné typu *LiveData*, které vrstva *View* pozoruje a na jejich změny okamžitě reaguje.

V mé aplikaci je grafické rozhraní vytvořeno zvlášť v oddělených XML[4] souborech které jsou potom s fragmentem propojeny.

ViewModel slouží jako prostředník mezi vrstvami *View* a *Model*. Připravuje data od vrstvy *Model* pro vrstvu *View*. Vrstva *ViewModel* obsahuje referenci na vrstvu *Model* ale neobsahuje žádnou referenci na vrstvu *View*, takže spolu nekomunikují přímo ale přes pozorovatelné proměnné typu *LiveData* a nebo *MutableLiveData*[5].



Obrázek 3: *Diagram MVVM*

[3]Knihovna Room https://developer.android.com/topic/libraries/architecture/room
[4]XML: Extensible Markup Language
[5]LiveData a MutableLiveData https://developer.android.com/topic/libraries/architecture/livedata

2.4 Komponenty a knihovny

Zde popisuji komponenty ze sady *Android JetPack*[6] a knihovny, které v aplikaci používám.

Sada Android JetPack

Android JetPack je sada knihoven, nástrojů a sbírka nejlepších postupů doporučených *Android týmem*. Jejích uživatelům pomáhají urychlit a zjednodušit psaní komplexních aplikací. Komponenty z této sady jsou vytvořené tak, aby fungovaly společně ale můžou být použity také individuálně. Pro tvorbu této aplikace jsem použil komponenty na zprávu databází, posílání zpráv o událostech a navigaci mezi fragmenty. Implementace těchto komponent byla jednoduchá, jediné co bych *Android* týmu vytkl je to, že ukázky v dokumentaci jsou psané převážně v jazyku *Kotlin*[7] místo *Javy*.

LiveData a MutableLiveData

Oproti normálním pozorovatelným proměnným *LiveData* jsou si vědoma životního cyklu fragmentů. Díky tomu neoznamují změny objektům, které nejsou aktivní a zvládají změny stavu jako například otočení obrazovky. *MutableLiveData* se na rozdíl od *LiveData* mohou měnit i po inicializaci.

Obrázek 4: *Android Jetpack Logo*

[6] Android Jetpack - https://developer.android.com/jetpack
[7] Kotlin - moderní programovací jazyk od firmy JetBrains https://kotlinlang.org/

2.4.1 SQLite a Room

SQLite je relační databáze, která nekomunikuje se vzdáleným serverem, ale je přiložená přímo v aplikaci. Dřív se v *Androidu* používala s knihovnou *SQLiteOpenHelper*, ale firma *Google* jí nahradila moderní knihovnou nazvanou *Room*.

Knihovna *Room* je abstrakce nad *SQLite* databází která zjednodušuje její používání a je navrhnuta tak, že spolupracuje s pozorovatelnými proměnnými typu *LiveData*.

Skládá se ze tří části.

- *Database* je abstraktní třída, která obsahuje instanci *SQLite* databáze, seznam *Entit* a *Data Access Object.*
- *Entity* je třída, která modeluje tabulku v databázi. Obsahuje *GET* a *SET* metody pro sloupečky v tabulce.
- *Data Access Object* je *interface*, přes který se přistupuje k datům v databázi. Definuje v sobě metody pro vkládání, čtení a mazání dat pomocí *SQL* příkazů.

```
1  @Fts4
2  @Entity(tableName = "fts_books_names")
3  public class ItemFts {
4     @ColumnInfo(name = "ean")
5     private String mEan;
6     @ColumnInfo(name = "name")
7     private String mName;
8     @ColumnInfo(name = "normalizedName")
9     private String mNormalizedName;
10
11    public ItemFts(String ean, String name, String normalizedName) {
12       this.mEan = ean;
13       this.mName = name;
14       this.mNormalizedName = normalizedName;
15    }
16    public String getEan() {
17       return this.mEan;
18    }
19    public String getName() {
20       return this.mName;
21    }
22    public String getNormalizedName() {
23       return this.mNormalizedName;
24    }
25 }
```

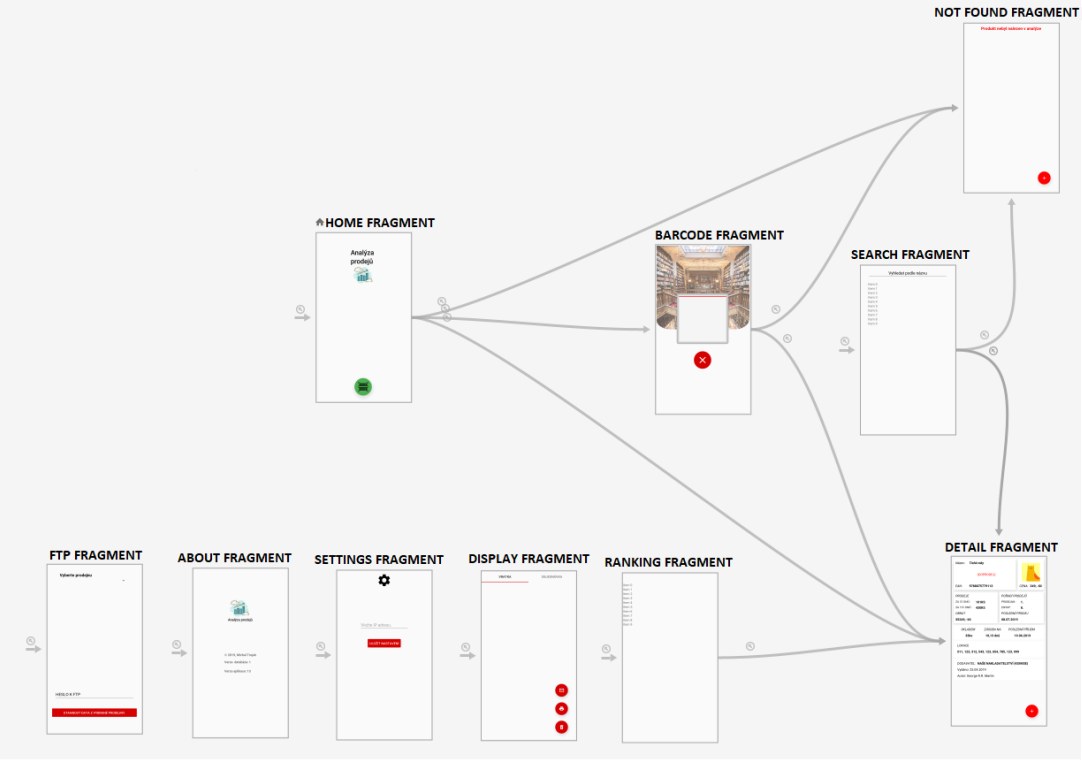
Zdrojový kód 1: *Příklad entity pro fulltextové vyhledávání*

2.4.2 Navigační komponent

Je součástí sady *Android Jetpack*. Poskytuje řešení pro všechny problémy spojené s navigaci mezi fragmenty v aplikaci a umožňuje vizualizaci přechodů mezi fragmenty.

Skládá ze čtyř částí.

- *Host* je vstupní aktivita aplikace, do které se vkládají fragmenty.
- *Navigation graph* obsahuje destinace a akce. Vytváří takzvanou *StoryBoard*, která poskytuje vyšší pohled na navigaci mezi fragmenty.
- *Destinations* jsou jednotlivé fragmenty aplikace.
- *Action* reprezentuje přechod mezi fragmenty. Specifikuji odkud kam. Nastavují se zde přechodové animace a to, do kterého fragmentu se aplikace přesune po kliknutí na tlačítko zpět.

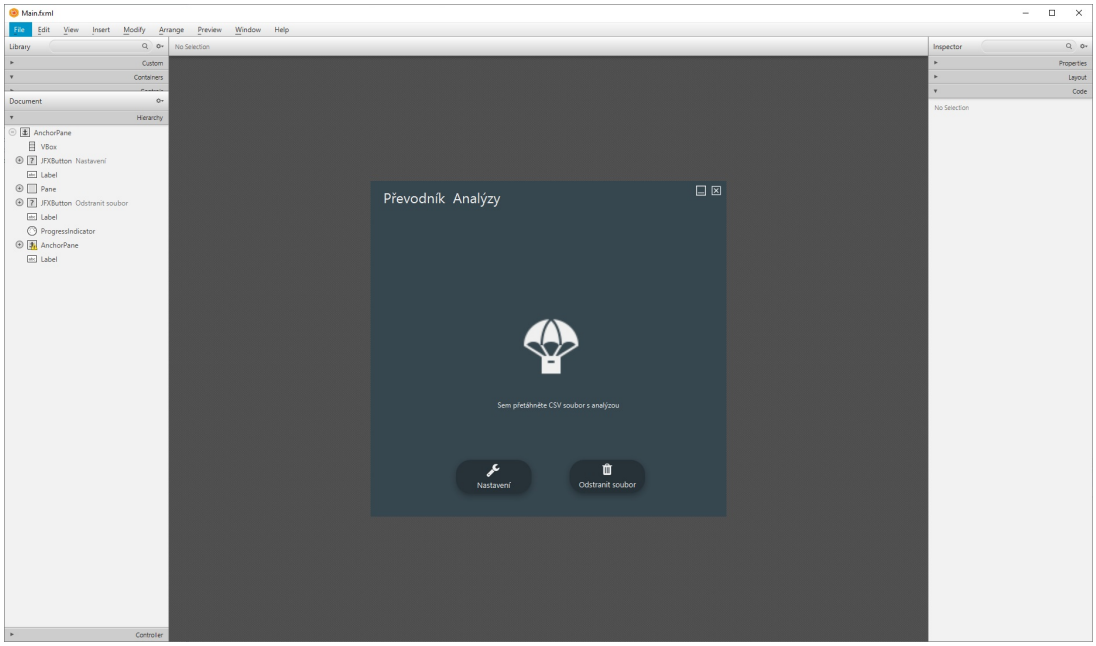


Obrázek 5: *Navigační graf*

2.5 JavaFX

Pro tvorbu grafického rozhraní podpůrného programu jsem se rozhodl použít platformu *JavaFX*. Hlavním důvodem byla jednoduchost nástroje *SceneBuilder*[8] od firmy *Gluon*[9].

SceneBuilder vygeneruje grafické rozhraní do FXML[10] souboru. Díky tomu se jednoduše implementuje architektura MVC[11], která je podobná výše popsané architektuře MVVM. S tím rozdílem, že vrstva *ViewModel* je nahrazena vrstvou *Controller*, která obsahuje reference na vrstvu *Model* a *View*.



Obrázek 6: *JavaFX SceneBuilder*

[8] SceneBuilder https://gluonhq.com/products/scene-builder/
[9] Gluon https://gluonhq.com/
[10] FXML - značkovací jazyk pro JavaFX
[11] MVC : Model-View-Controller

2.6 Transportní protokoly

Jedním z hlavních požadavku na tuto aplikaci je schopnost přeposílat data mezi mobilní aplikací a podpůrným programem. K přeposílání dat mezi dvěma aplikacemi se používají transportní protokoly.

V této kapitolo tyto protokoly krátce popíšu a vysvětlím, proč jsem pro implementaci vybral ten, který jsem vybral.

User Datagram Protocol

Je to protokol, který poskytuje nespojovanou a nespolehlivou službu. Nespojovaná služba znamená, že odesílatel a příjemce mezi sebou před odesláním dat nevytváří spojení a nespolehlivá služba znamená, že protokol nezaručuje doručení dat.

Výhody tohoto protokolu oproti TCP protokolu je právě to, že odesílatel a příjemce nevytváří spojení a je tedy rychlejší, a dále také možnost odesílat data od jednoho odesilatele pro celou skupinu příjemců najednou. Hlavní nevýhoda UDP protokolu a hlavní důvod proč jsem ho v implementaci nevyužil je to, že negarantuje doručení dat, takže by v případě nedoručení paketu paket znova nezaslal a to by pro potřebu této aplikace byl zásadní problém.

Transmission Control Protocol

Tento protokol poskytuje spojovanou a spolehlivou službu. Spojovaná služba znamená, že předtím než se začnou data odesílat, tak odesílatel a příjemce mezi sebou vytvoří spojení. Spolehlivá znamená to, že TCP protokol garantuje doručení dat. V případě, kdy odesílatel nedostane po určitém čase od příjemce potvrzení o doručení paketu, tak paket znova odešle.

Po porovnání těchto dvou protokolu a zvážení potřeb aplikace jsem došel k závěru, že spolehlivost a správnou dat je důležitější než rychlost a proto jsem k implementaci přeposílání dat mezi aplikací a pomocným programem zvolil protokol TCP.

3 Analýza požadavků

V této části krátce vysvětlím co je a k čemu slouží proces specifikace požadavku a co si představit pod pojmem požadavek. Také zde vytvořím diagram případů užití a sestavím seznam funkčních a nefunkčních požadavků pro hlavní aplikaci a podpůrný program.

3.1 Proces specifikace požadavků

Sestavování softwarových požadavků je proces, který se používá při tvorbě softwarových produktů. Je to proces tvorby dokumentu, ve kterém jsou zapsány všechny požadavky zadavatele, podle kterých se poté software vytváří. Důsledné definování požadavků pro výsledný produkt je klíčové, protože usnadňuje následnou komunikaci mezi zadavatelem a programátorem.

Požadavek

Samotný požadavek je vlastnost a nebo schopnost kterou zadavatel chce aby konečný softwarový produkt splňoval.

Ve standardizovaném slovníku terminologie softwarového inženýrství Institutu pro elektrotechnické a elektronické inženýrství [IEEE, 24765-2017][1] je požadavek definován jako:

1. Podmínka nebo schopnost nutná k dosažení cíle a nebo k vyřešení problému uživatele.
2. Podmínka nebo schopnost, která musí být splněna a nebo musí být obsažena v systému či jeho součásti. Tak aby to vyhovovalo smlouvě, standardu a nebo jinému vztahujícímu se dokumentu.
3. Zdokumentována reprezentace splnění podmínek nebo schopností z bodu 1 a 2.

Rozdělení požadavků

Požadavky se mohou dělit různě, pro svojí potřebu jsem zvolil rozdělení na funkční požadavky a mimo funkční požadavky.

- Funkční požadavky jsou požadavky na systém a nebo jeho komponentu, které přesně vyjadřují co má konečný softwarový produkt dělat. Popisují funkce a služby, které jsou od softwaru očekávány.
- Nefunkční požadavky jsou požadavky, které se netýkají funkcí systému, ale jeho vlastností.

3.2 Požadavky pro hlavní aplikaci

Funkční požadavky

- Aplikace musí uživateli umožnit importování dat z analýzy prodejů libovolné prodejny.
- Aplikace musí být schopná importovat data přes lokální sít z počítače pomocí podpůrného programu.
- Aplikace musí uživateli nabídnou možnost výběru prodejny a stažení dat z FTP serveru.
- Pokud je k dispozici bezdrátové připojení k internetu, musí aplikace umožnit automatickou synchronizaci dat z FTP serveru pro zvolenou prodejnu.
- Aplikace musí uživateli umožnit vyhledávání produktů v analýze prodejů
  - pomocí čtečky čárového kódu,
  - ručním zadáním čárového kódu,
  - a podle názvu.
- V případě, kdy analýza prodejů obsahuje hledanou knížku, tak aplikace musí zobrazit detail knížky, ve kterém musí být zobrazeny klíčové údaje o knížce:
  - název knihy,
  - čárový kód EAN[12],
  - doporučenou prodejní cenu,
  - obrázek titulní strany (pokud je dostupný),
  - celkový počet prodejů za posledních 31 dní a za celé účetní období,
  - celkový obrat v Českých korunách,
  - pořadí prodejů na prodejně a v e-shopu,
  - datum posledního prodeje a příjmu na sklad,
  - kolik kusů je na prodejně skladem a na kolik dní tato zásoba vydrží,
  - název dodavatele,
  - lokace na kterých je zboží na prodejně umístěno,
  - datum vydání knížky,
  - a autory knížky.

[12]EAN je typ čárového kódu používaný k označení zboží v Evropě

- V opačném případě stačí uživatele informovat o tom, že ke knížce nejsou dostupné informace.
- Aplikace musí umožnit uživateli vytvořit seznam zboží na vratku a nebo objednávku.
- Aplikace musí umožnit uživateli exportovat seznam zboží do počítače v excelovém formátu.
- Aplikace musí umožnit uživateli tisknout seznam zboží přes WIFI tiskárnu. Vytisknuté položky musí obsahovat EAN, název, počet a umístění zboží na prodejně.
- Aplikace musí umožnit uživateli zobrazit žebříček prodejů.

Nefunkční požadavky

- Aplikace musí být vytvořena pro operační systém *Android*.
- Aplikace musí být naprogramovaná v programovacím jazyce *Java*.
- Aplikace musí podporovat různé velikosti obrazovek mobilních telefonu.
- Aplikace musí mít jednoduché a přehledné grafické rozhraní.
- Aplikace musí ukládat data lokálně v *SQLite* databázi.

3.3 Požadavky pro podpůrný program

Funkční požadavky

- Program musí obsahovat funkci pro nahrávání dat z CSV[13] souboru do programu.
- Program musí přečíst data ze souboru typu CSV a uložit je do kolekce objektů (hašovací mapy).
- Program musí být schopný převézt data z kolekce do JSON[14] objektu a na požádání klienta (hlavní aplikace) je odeslat.
- Program musí být schopný přijímat data od klienta a uložit je do souboru v excelovém formátu.
- Program musí umožnit uživateli výběr složky, do které se uložit vyexportovaná data od klienta.

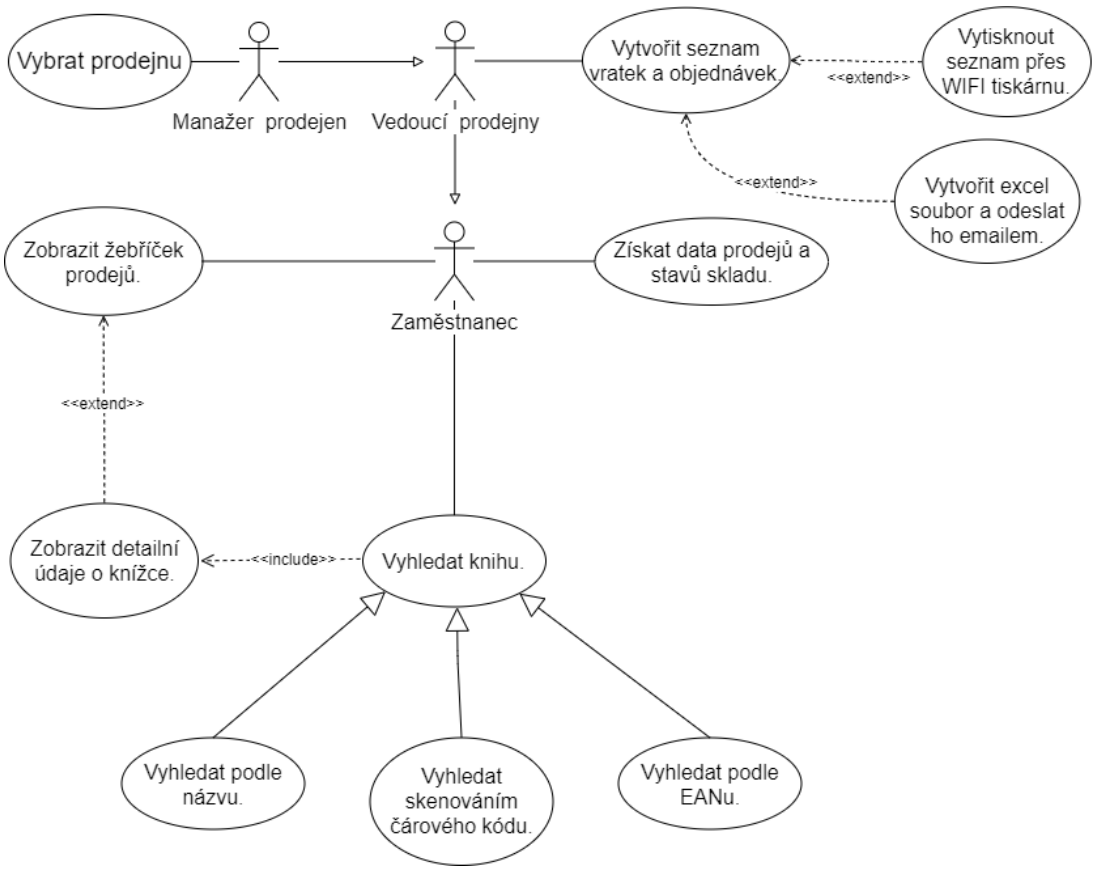
[13] Comma-Separated Values
[14] JSON - Javascript Object Notation, je formát pro výměnu dat

Nefunkční požadavky

- Program musí být vytvořen pro operační systém *Windows*.
- Program musí být naprogramovaný v programovacím jazyce *Java*.
- Program musí mít grafické rozhraní naprogramované v *JavaFX*.

3.4 Diagram případů užití

Diagram popisuje funkcionalitu systému a zobrazuje chování systému z pohledu uživatele.



Obrázek 7: *Diagram případů užití*

4 Implementace mobilní aplikace

V této kapitole popisuji některé aktivity, fragment a jiné třídy a jejich metody, které dohromady tvoří hlavní aplikaci.

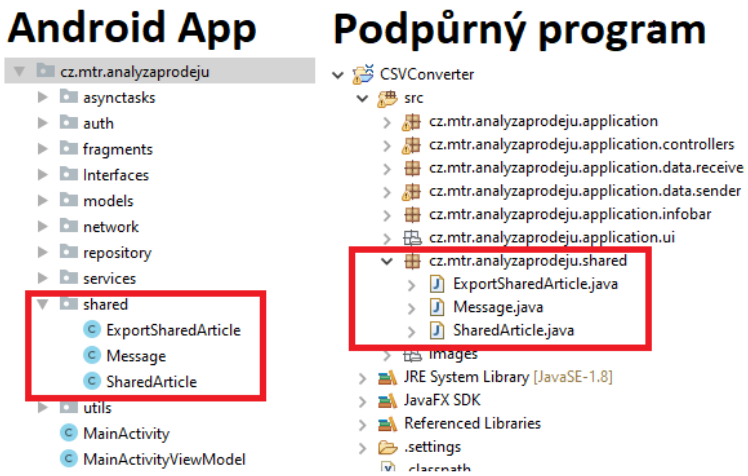
4.1 Article, SharedArticle, ExportArticle

V tomto textu mluvím o knihách ale v kódu pro ně používám obecný výraz pro zboží "*Article*".

Jejich zde několik druhů a při psaní této kapitoly jsem si uvědomil, že to může být matoucí a proto jsem se rozhodl jim věnovat tuto podkapitolu ve které je všechny popíšu.

- *Article* je třída, která reprezentuje *entitu* (Kap. 2.4.1) uloženou v databázi.
- *DisplayableArticle* je třída, která nese informace o knize a používá se při zobrazování ve fragmentu *DetailFragment*.
- *SharedArticle* je třída, která reprezentuje objekt přenášený při přenosu dat analýzy prodejů z podpůrného programů do mobilní aplikace (Kap. 6.2). Obsahuje informace o jedné knize (název, cena, množství prodaných kusu, dodavatel atp.).
- *ExportArticle* je to podtřída třídy *SharedArticle*, která je rozšířená o proměnné které nesou množství zboží na vrácení a objednání. Používá se při exportování dat z mobilní aplikace do podpůrného programu a následné tvorbě excelových souborů (Obr. 15).

SharedArticle a ExportArticle musí mít v obou programech stejný název a adresářovou strukturu jinak by se přenos dat TCP protokolem nezdařil.



Obrázek 8: *Stromy adresářů*

4.2 Aktivity

Při implementaci jsem se snažil dodržovat styl vývoje aplikací doporučený *Android* týmem. Aplikace tedy obsahuje pouze jednu aktivitu zvanou *MainActivity*, která slouží jako vstupní aktivita (Obr. 2) a také se v ní střídají fragmenty, které se mají zobrazit.

- Je zde hlavní menu, kterému se říká *Hamburger menu*, implementované pomocí *DrawerLayout* a obsluha tlačítek v hlavním menu.
- *NavController*, který má na starostí přepínání fragmentu a *NavHost*, který slouží jako kontejner pro fragmenty.
- Provádí se tu inicializace *SharedPreferences*, ve kterých jsou uložené data, která je potřeba obnovit když se aplikace znova spustí.
- Spouští se zde služba na pozadí, které má na starosti automatickou aktualizaci dat.
- Kontroluje se zde, zda uživatel dal aplikaci povolení k používání Internetu, přístupu do interní paměti aj. (Kap. 4.6).
- Při prvním spuštění se tu inicializuje přiložená *SQLite databáze*.

4.2.1 Metody životního cyklu aktivity

Jsou to metody, které volá system. Není vždy potřeba implementovat všechny metody životního cyklu aktivity. V této aktivitě používám jen základní tři.

- *void onCreate()* je metoda, která se zavolá při prvním startu aplikace, inicializuje se v ní grafické rozhraní a vše co jsem popsal výše v kapitole 4.2.
- *void onStop()* je metoda, která se volá při přepnutí aplikace do pozadí. Tady se ukládají data, která budeme chtít obnovit až se aplikace vrátí do popředí. Po zavolání této metody aplikace přestane být viditelná.
- *void onStart()* je metoda, která se volá při přepnutí aplikace do popředí. Zde se obnovují data, která byla uložena při přepnutí aplikace do pozadí. Po zavolání této metody se aplikace stane viditelná.

4.3 Fragmenty

V této kapitole popisuji jednotlivé fragmenty (Kap. 2.2). Každý fragment volá *onCreate()* metodu, ve které inicializuje prvky svého grafického rozhraní.

Snažil jsem se dodržovat MVVM architekturu (Kap. 2.3), takže každý fragment data jenom zobrazuje, anebo je od uživatele přijímá a předává do své třídy *ViewModel*, která s nimi dále pracuje.

4.3.1 HomeFragment

Je to výchozí fragment, který umožňuje uživateli zahájit proces vyhledávání knih.

Implementuje se zde spodní lišta typu *BottomAppBar* která obsahuje tři tlačítka.

- Tlačítko typu *FloatingActionButton*, které spouští *BarcodeFragment* na vyhledávání knihy pomocí skenování čárového kódu.
- A dvě tlačítka typu *ImageButton* pro vyhledávání podle názvu a manuálním zadáním čárového kódu[15].

HomeViewModel inicializuje třídu SearchByEan a ta obsahuje metodu *findArticle(String ean)*.

Metoda *findArticle(String ean)* vyhledává knihu v datové struktuře typu *HashMap*, ve které jsou uloženy data z analýzy prodejů. Pokud jí tam nenajde, tak se zkusí podívat i do přiložené databáze.

Pro datovou strukturu typu *HashMap* jsem se rozhodl kvůli tomu, že má konstantní časovou náročnost při operaci vyhledávání bez ohledu na počet uložených objektů.

Během vyhledávání knihy můžou nastat dvě situace.

- Čárový kód byl v analýze nalezen. Zavolá se metoda *goToDetailFragment()* a přejde se do fragmentu *DetailFragment*, kterému se jako parametr předá objekt třídy *SharedArticle*, ve kterém jsou informace o nalezené knize.
- Čárovy kód nebyl v analýze prodejů nalezen a zobrazí se zpráva "Čárový kód neexistuje".

4.3.2 BarcodeFragment

V tomto fragmentu (Obr. 13) se implementuje API *Mobile Vision*[16] od firmy *Google*, která umožňuje skenování čárových kódu. Podle výsledků vyhledání se chová stejně tak, jak je popsáno v části *HomeFragmentViewModel* (Kap. 4.3.1).

Je tu také tlačítko typu *FloatingActionButton*, které skenování zastaví a vrátí nás zpět do výchozího fragmentu.

[15] U knih je čárový kód stejný jako ISBN

[16] MobileVision API https://developers.google.com/vision/android/barcodes-overview

4.3.3 SearchFragment

Tento fragment má na starosti vyhledávání knih podle názvu. Vstupní informace od uživatele předává třídě *SearchViewModel*, která obsahuje databázi *DataRepository* pro fulltextové vyhledávání, ve které je tabulka se třemi sloupečky.

První sloupeček obsahuje název knihy, druhý sloupeček obsahuje čárový kód knihy a třetí sloupeček obsahuje normalizovaný název. Třetí sloupeček je indexovaný pro zrychlení fulltextové vyhledávání.

Normalizovaný název je stejný jako název, s tím rozdílem, že je ošetřený tak, že všechny písmena jsou v *lowercase* a bez diakritiky. Vstupní informace od uživatele jsou také ošetřeny, takže nezáleží na tom jestli při vyhledávání používá diakritiku a nebo ne.

Podle výsledků vyhledání se chová stejně tak, jak je popsáno v části *HomeViewModel* (Kap. 4.3.1).

4.3.4 DetailFragment

Tento fragment (Obr. 13) má dvě funkce. První funkce je zobrazování informací o nalezené knize uložené v objektu typu *DisplayableArticle* a druhá funkce umožňuje vkládání knih do seznamu vratek a nebo objednávek.

Implementaci těchto funkci nemá na starosti třída *DetailFragment* ale třída *DetailViewModel*, protože *DetailFragment* slouží pouze k zobrazování informací o knize a jako místo pro zadání vstupní informace o tom, kolik knížek chce uživatel vrátit a nebo objednat.

DetailViewModel obsahuje instanci databáze *LinkRepository* která obsahuje odkazy na obrázek titulní strany vybrané knihy a několik metod:

- *getArticle()* přes kterou *DetailFragment* získá informace o knize k zobrazení,
- *loadImage()* které podle objektu vybere správny odkaz a předá ho fragmentu *DetailFragment* aby zobrazil titulní stranu,
- *saveArticleAndAmountOrders()* kterou *DetailFragment* používá pro vložení knihy do seznamu objednávky,
- *saveArticleAndAmountReturns()* kterou *DetailFragment* používá pro vložení knihy do seznamu vratky.

Seznamy jsou implementovány pomocí kolekcí typu *ArrayList<ExportSharedArticle>*, které jsou ve statické datové třídě *Model*, kterou *DetailViewModel* používá.

4.3.5 DisplayFragment

Tento fragment slouží k zobrazení seznamů vratek a objednávek (Obr. 14). A jsou tu tři tlačítka typu *FloatingActionButton* umožnující provádění operací se seznamy. Je zde také implementováno několik grafických ovládacích prvky.

- *ViewPager* je kontejner, ve kterém jsou dva uloženy fragmenty uvnitř posuvných záložek.
- *ReturnsFragment* a *OrdersFragment* jsou fragmenty uložené v kontejneru *ViewPager*. Jsou v nich zobrazené seznamy vratek a objednávek na které je možné klikat a upravovat množství.

DisplayViewModel obsahuje dvě metody na obsluhu tlačítek ve fragmentu *DisplayFragment*.

- *createFileAndOpenEmailClient()* vytvoří excelové soubory se seznamy vratek a objednávek a přiloží je jako přílohu do emailu.
- *print()* vytvoří ze seznamu vratek a objednávek *printJob* a odešle ho do tiskové fronty vybrané tiskárny.

4.3.6 FtpFragment

V tomto fragmentu se zadává heslo k FTP serveru a vybírá prodejna, ze které se má stáhnout analýza prodejů. Jsou zde také tři grafické ovládací prvky.

- Tlačítko typu *Button* na spuštění stahování.
- Rozbalovací seznam typu *Spinner*, ve kterém jsou předdefinované prodejny k výběru.
- Textové pole typu *TextInputEditText*, do kterého se zadává heslo.

FtpViewModel je třída, která obsahuje několik metod, které se používají ke stahování dat z FTP serveru.

- *onDownloadDataClick()* je metoda, která po kliknutí na tlačítko pro spuštění stahování zavolá metody *setSelectedStore()*, *setPassword()* a vytvoří nové vlákno, ve kterém se zahájí stahování.
- *setSelectedStore()* uloží do lokálního nastavení typu *SharedPreferences* prodejnu vybranou z rozbalovacího seznamu.
- *setPassword()* uloží do lokálního nastavení typu *SharedPreferences* heslo zadané do textového pole.

4.3.7 RankingFragment

Tento fragment zobrazuje žebříček prodejů knih (Obr. 12). Pokud se na nějakou knihu v žebříčku klikne, tak se otevře detail té knihy. Implementují se tu jen dva grafické ovládací prvky.

- *TextView*, ve kterém se zobrazují datum a celkový obrat.
- *RecyclerView* zobrazuje seznam, který má určitý počet buněk, které se neustále recyklují k zobrazení dat. Data se před zobrazením nahrají do buňky a po zobrazení se z té buňky odstraní aby byla buňka dostupná pro další data, která potřebují zobrazit. Tím se zabraňuje zbytečnému vytváření buněk pro data, které nejsou zobrazena.

RankingViewModel obsahuje instanci třídy typu *RankingRepository* a v ní pozorovatelný list typu *List<RankingItem>* se sestupně seřazenými daty od nejprodávanější knihy po nejméně prodávanou knihu.

Jsou tu dvě metody:

- *getAllItems()*, která fragmentu předává k zobrazení žebříček prodejů,
- *getArticleFromAnalysis()*, která zobrazuje detail knihy po kliknutím na položku v žebříčku.

4.4 Třídy AsyncTask

Grafické rozhraní v operačním systému *Android* běží na hlavním vlákně. Pokud se na hlavním vlákně spustí nějaká náročnější operace tak grafické rozhraní zamrzne. Toto chování se obchází pomocí třídy typu *AsyncTask* která pro náročnou operaci vytvoří nové vlákno.

Všechny třídy, které z této třídy dědí musí implementovat několik metod:

- *onPreExecute()* do této metody se dává kód, který se má vykonat před spuštěním nového vlákna. V mé aplikaci se tu zobrazuje dialogové okno s ukazatelem průběhu typu *ProgressBar*,
- *doInBackground()* zde se provádí hlavní operace,
- *onProgressUpdate()* zde se může průběžně aktualizovat ukazatel průběhu podle aktuálního stavu operace,
- *onPostExecute()* zde je kód, který se provede po dokončení hlavní operace. V mé aplikaci se zde ukládají data a schovává dialogové okno s ukazatelem průběhu typu *ProgressBar*.

V aplikaci je několik tříd typu *AsyncTask*.

- **UnziperTask** při prvním spuštění rozbalí přiloženou *SQLite* databázi.
- **DownloadAnalysisFtpTask** stahuje analýzy prodejů z FTP serveru.
- **DownloadStoreDataFtpTask** stahuje data o stavech skladů na prodejnách z FTP serveru.
- **UpdateAnalysisTask** má na starosti automatickou aktualizaci dat z analýzy prodejů.
- **UpdateStoreDataTask** má na starosti automatickou aktualizaci dat o stavech skladů na prodejně.

4.5 JobService

JobService je typ tříd reprezentující služby, které periodicky běží na pozadí. V aplikaci používám jen dvě služby, které jednou za hodinu zkontrolují, zda na FTP serveru došlo k aktualizaci dat, pokud ano tak spustí *AsyncTask*, který zahájí aktualizaci dat v aplikaci.

- **UpdateAnalysisJobService** je služba kontrolující změny na FTP serveru s daty z analýzy prodejů vybrané prodejny.
- **UpdateAnalysisStoreDataService** je služba kontrolující změny na FTP serveru s daty o stavech skladů na prodejně

4.6 Povolení

V zájmu ochrany soukromý uživatele si aplikace musí za běhu jednotlivě požádat o povolení pokud chce získat přistup k citlivým datům uživatele a nebo systémovým funkcím jako je kamera a nebo Internet. [Android dokumentace][3].

Tato aplikace žádá o tři povolení.

- Kvůli skenování čárových kódů potřebuje povolení k použití kamery.
- Pro tvorbu excelových souborů se seznamem vratek a objednávek potřebuje povolení ke čtení a psaní do interní paměti.
- Kvůli stahování analýz prodejů a aktualizaci dat potřebuje povolní k Internetu.

5 Implementace pomocného programu

V této kapitole popisuji implementaci tříd pomocného programu. Snažil jsem se dodržovat architekturu MVC.[17]

5.1 Grafické rozhraní

Grafické rozhraní je psané na platformě *JavaFX*(Kap. 2.5). Na výběr jsem měl dvě možnosti. Bud to napsat přímo v *Java* kódu a nebo použít nástroj *SceneBuilder* od firmy *Gluen*, ve kterém je možné navrhovat uživatelské rozhraní stylem "táhni a pusť" a který následně vygeneruje soubor ve formátu *FXML*.

Zvolil jsem druhou možnost, protože jsem chtěl mít grafické rozhraní úplně oddělené od kódu aplikace a byla to příležitost naučit se *FXML*.

Soubor *Main.fxml* je soubor, obsahující vygenerovaný *FXML* kód grafického rozhraní z programu *SceneBuilder*. Má stromovou strukturu, ve které je rodičem třída typu *AnchorPane*. A v ní jsou uloženy potomci ve formě komponentů, které mají nějaké ovládací funkce. Používám jich tu hned několik.

- **JFXButton** je komponent z knihovny *JFoenix*[18], který vytváří moderně vypadající tlačítka.
- **Label** je komponent, ve kterém se zobrazuje informace a tom, co program právě dělá.
- **ProgressIndicatior** je komponent, který zobrazuje, že program pracuje.
- **ImageView** je komponent, ve kterém je uložený obrázek padáku. Tento obrázek označuje místo, na které má uživatel přetáhnout soubor s daty pomocí funkce "táhni a pusť".

Funkce "táhni a pusť"

Implementování této funkce je v *JavaFX* velmi jednoduché. Na *ImageView* komponent stačilo přidat dva posluchače události.

- *onDragOver*, který posílá zprávu když je tažený objekt nad vybranou oblastí.
- *onDragDropped*, který odešle zprávu když uživatel soubor nad vybranou oblastí upustí. Tato zpráva obsahuje absolutní cestu k souboru s daty a je předávána metodě *handleDrop()* ve třídě **Controller**.

[17] MVC Model-View-Controller
[18] JFoenix - knihovna moderních komponentu pro JavaFX

5.2 Controller

V této třídě dochází ke spojení mezi grafickým rozhraním a byznys logikou.

- *handleDrop()* od grafického rozhraní přijímá absolutní cestu k souboru s daty a volá metodu *readAnalysisFile()*.
- *readAnalysisFile()* otevře CSV soubor s daty a data převede do hašovací mapy a tu uloží ve třídě *Model*.

Také se tu inicializují třídy Sender a Receiver, které vytváří *TCP* spojení (Kap. 2.6) s hlavní aplikací.

Sender

Tato třída vytváří spojení mezi hlavní aplikaci a pomocným program. Je zde metoda *listen()* která naslouchá k portu číslo 9998. Jakmile si na tomto portu hlavní aplikace zažádá o data z analýzy prodejů. Tak se vytvoří instance třídy *Message* do které se data zabalí a tento objekt se odešle do hlavní aplikace.

Message

Tato třída se používá pro přesun dat po síti. Implementuje rozhraní *Serializable* a díky tomu je možné data převést na sekvenci bitu a po doručení je zase převést do původního stavu.

Receiver

Podobně jako třída *Sender* slouží ke komunikaci s hlavní aplikací. Je zde metoda *listen()* které naslouchá portu 5678. Na tomto portu od hlavní aplikace příjme data o vratkách a objednávkách. Po příjetí vytvoří instanci třídy *FileCreator* a zavolá její metodu *createFiles()*.

FileCreator

Tato třída má na starosti vytvoření excelových souborů s daty o vratkách a objednávkách. K jejich tvorbě používá knihovnu *Apache POI*[19].

[19] Apache POI - knihovna pro tvorbu excelových souborů

5.3 Model

Tato třída je *Singleton*[20] která slouží jako "božská třída" přes kterou se nahrávají a ukládají data a také je zde inicializovaná třída *Settings*.

Settings

V této třídě implementuji *Java Preferences Api*, která vytváří konfigurační soubor, ve kterém je uložena absolutní cesta ke složce, do které se budou ukládat excelové soubory přijaté z hlavní aplikace. Jako výchozí je nastavená ta složka, ze které byl program spuštěn.

[20]Singleton - třída, která může být inicializovaná jenom jednou

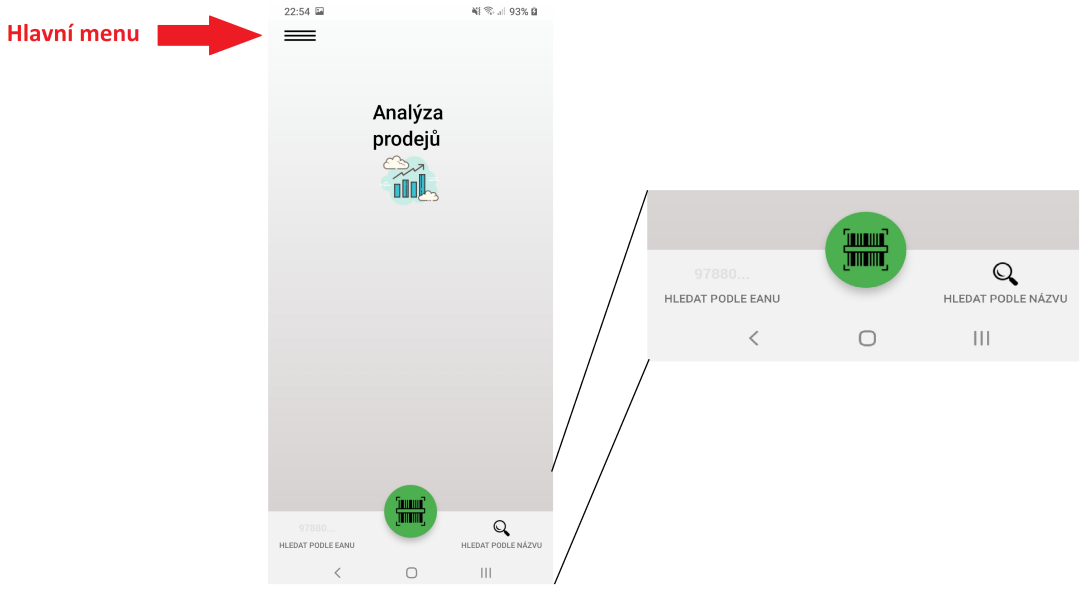
6 Uživatelská příručka

V této kapitole popisuji použití aplikace. Jak se získávají data z analýzy prodejů, jak funguje vyhledávaní knížky a nakonec i jak vytváří vratka a objednávka.

6.1 První spuštění

Během prvního spuštění Vás aplikace nejdřív požádá o udělení oprávnění ke čtení a ukládání dat z interní paměti zařízení. Tato oprávnění jsou nutná pro tvorbu excelových souborů se seznamem zboží na vrácení a objednání.

Zároveň se během prvního spuštění rozbalí přiložená *SQLite* databáze s čárovými kódy a názvy všech knížek, které firma Knihy Dobrovský prodává. Tato databáze se používá k fulltextovém vyhledávání podle názvu knížky a obsahuje přes 450 000 záznamů, takže v závislosti na výkonu Vašeho zařízení může první spuštění trvat o něco déle.



Obrázek 9: *Hlavní aktivita.*

6.2 Získávání dat z analýzy prodejů

V první verzi této aplikace bylo možné získat data z analýzy prodejů pouze jedním způsobem. Bylo nutné data vyexportovat z informačního systému firmy v CSV formátu, a poté pomocí podpůrného programu "prevodnik.jar" data převést do binární formy a přeposlat po síti do mobilní aplikace.

Tento způsob byl však zdlouhavý a po domluvě s vedením firmy byl vytvořen FTP server, na který se data z informačního systému automaticky nahrávají.

Po vybrání prodejny a zadání hesla k FTP serveru je aplikace schopna si je stáhnout a automaticky aktualizovat.

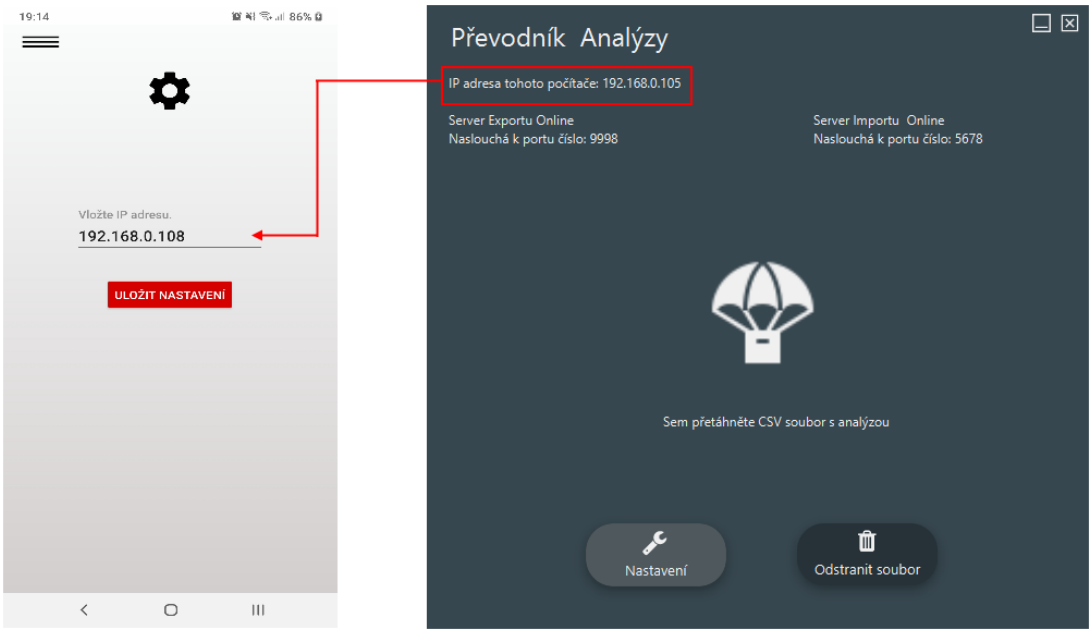
V této části popíšu oba způsoby získávání dat.

Importování dat pomocí podpůrného programu

Ke spuštění pomocného programu je nutné mít nainstalovanou Javu a spouští se souborem "prevodník.jar". V levém rohu se zobrazí IP adresa Vašeho počítače, která musí být nastavena v aplikaci. To se dělá v "Nastavení IP" v menu aplikace.

Jakmile je IP adresa nastavená, tak už jenom stačí přetáhnout CSV soubor s daty z analýzy prodejů na ikonu padáku a v menu aplikace zahájit přenos dat do aplikace pomocí tlačítka "Nahrát data z PC".

Po nahrání dat je aplikace připravená k použití.



Obrázek 10: *Propojení aplikace s podpůrným programem.*

Importování dat z FTP serveru

Tento způsob je mnohem jednoduší a pohodlnější než ten předchozí. V hlavním menu aplikace stačí kliknout na tlačítko "Nahrát data z FPT". Poté vybrat prodejnu, vložit heslo a data se stáhnou po kliknutí na tlačítko "Stáhnout data z vybrané prodejny".

V pozadí aplikace se zároveň spustí služba, která v případě změny dat na FTP serveru provede automatickou aktualizaci dat.

Některé prodejny ještě nepřešly na nový informační systém a proto jsou jejich údaje z analýzy prodejů nekompletní. V době psání tohoto textu byly nejlepší prodejny na testování Brno Joštovka a Brno Vaňkovka.

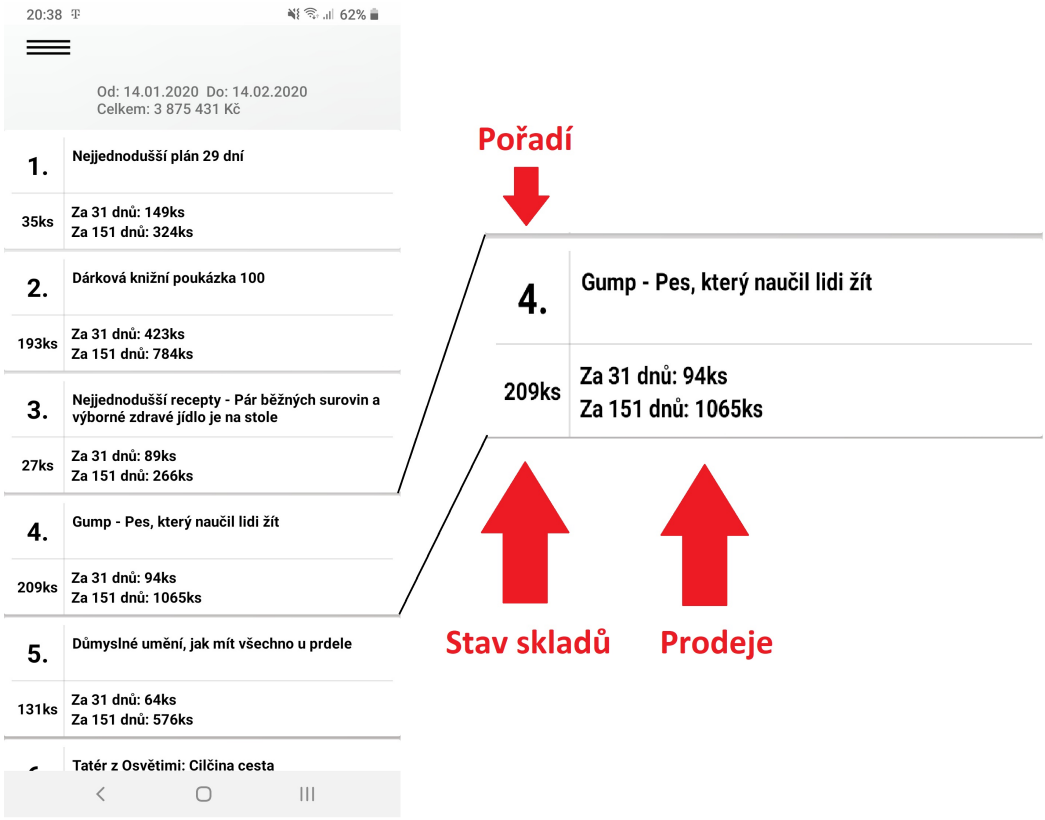
Obrázek 11: *Fragment pro stahování dat z FTP.*

6.3 Žebříček prodejů

Po nahrání dat z analýzy prodejů do aplikace se umožní zobrazování žebříčku prodejů pro danou prodejnu. Žebříček se zobrazí pomocí tlačítka "Žebříček prodejů" v menu aplikace.

Nahoře se zobrazuje celková tržba za poslední měsíc. U každé knížky jsou zobrazeny informace o pořadí prodejů v rámci prodejny, počtu kusů ve skladu, název, prodeje za poslední měsíc a od začátku účetního roku.

Po kliknutí na knížku se zobrazí detailní informace o knížce.

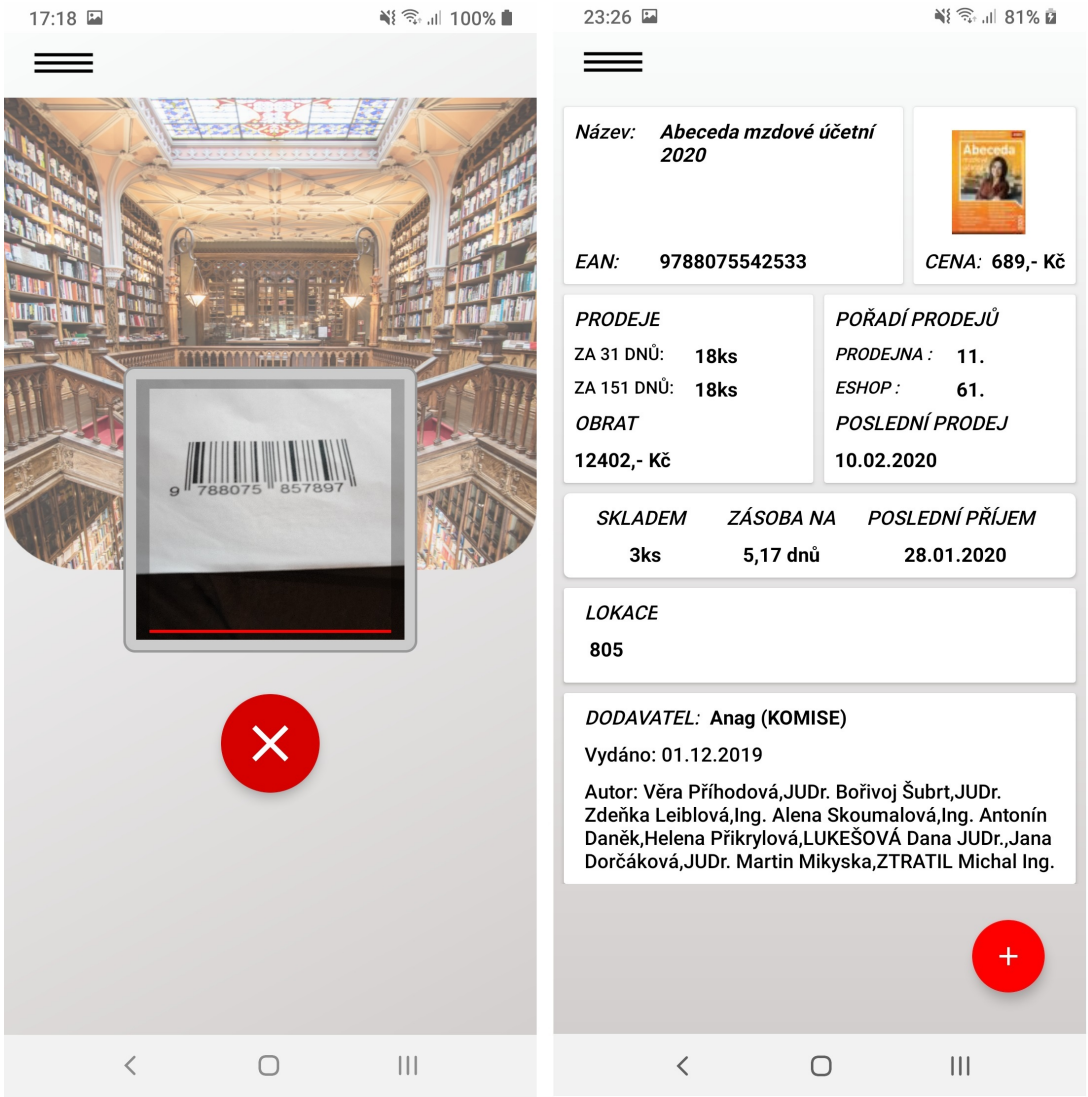


Obrázek 12: *Fragment žebříčku prodejů.*

6.4 Vyhledávání a detail knihy

Knihu je možné vyhledat podle názvu knihy, naskenováním čárového kódu a nebo ručním zadáním čárového kódu (Obr. 9). Po nalezení knihy se zobrazí fragment obsahující detailní informace o knize.

Z tohoto fragmentu můžete knihu přidat do vratky nebo do objednávky kliknutím červeného tlačítka se symbolem plus (Obr. 14).



Obrázek 13: *Fragment skenování čárových kódu a fragment detailu knihy.*

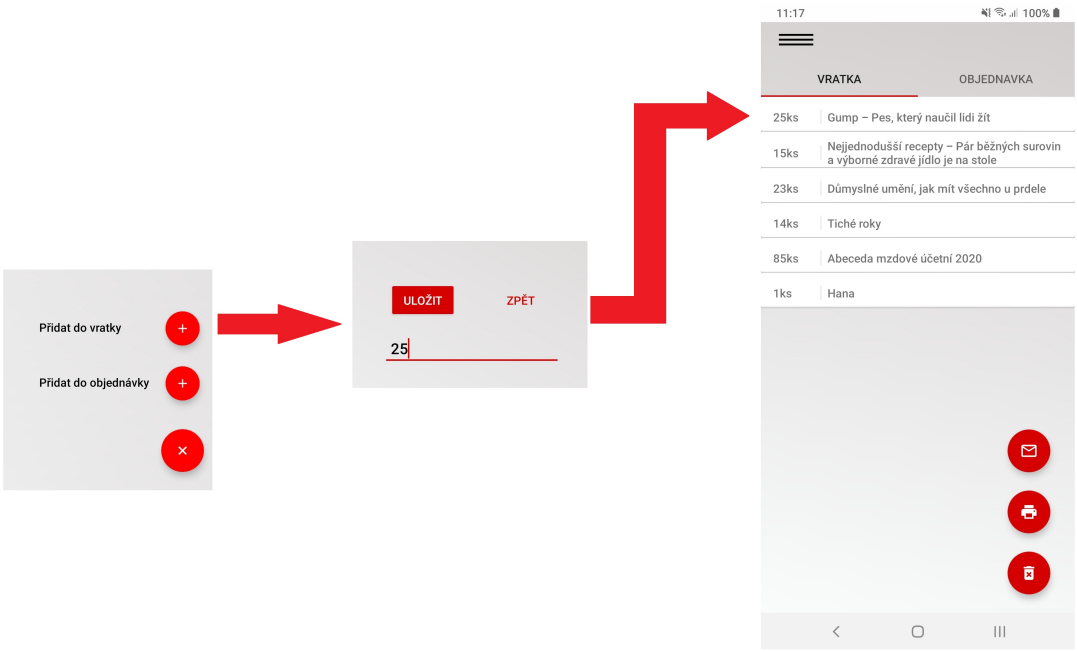
6.5 Vratky a objednávky

K tomuto fragmentu je možné se dostat přes hlavní menu kliknutím na "Vratky a objednávky". Jsou zde zobrazené všechny knihy, které chceme vrátit dodavateli a nebo které potřebujeme objednat. Najdeme tady i tři tlačítka.

Kliknutím tlačítka se symbolem obálky se otevře Váš emailový klient a do přílohy se vloží excelový soubor se seznamem knih (Obr. 15). Tlačítko se symbolem tiskárny Vám nabídne možnost vytisknout seznam knih v libovolné tiskárně na lokální síti.

Poslední tlačítko s symbolem odpadkového seznam vymaže celý seznam knih.

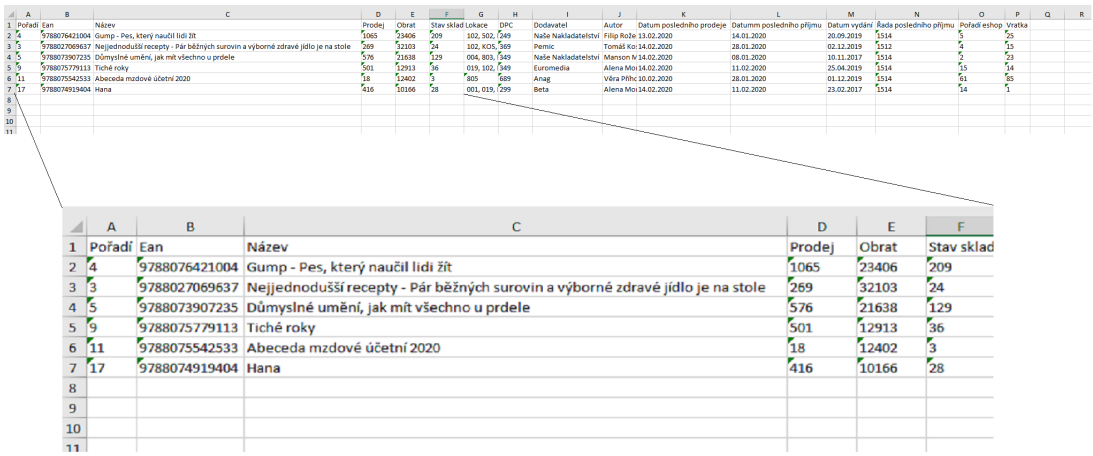
V seznamu knih je možné kliknutím na knihu zobrazit dialog pro změnu množství a nebo vymazání knihy ze seznamu.



Obrázek 14: *Příklad přidání knihy do vratky.*

6.6 Exportování seznamů vrátek a objednávek do počítače

K vyexportování excelových souborů do počítače stačí spustit podpůrný program, správně nastavit IP adresu (Obr. 10) a v menu aplikace kliknout na "Odeslat data do PC". Exportovaný soubor se vytvoří ve složce jejíž cestu je možné nastavit v nastavení podpůrného programu. Výchozí cesta je cesta k souboru "prevodník.jar".



	A	B	C	D	E	F
1	Pořadí	Ean	Název	Prodej	Obrat	Stav sklad
2	4	9788076421004	Gump - Pes, který naučil lidi žít	1065	23406	209
3	3	9788027069637	Nejjednodušší recepty - Pár běžných surovin a výborné zdravé jídlo je na stole	269	32103	24
4	5	9788073907235	Důmyslné umění, jak mít všechno u prdele	576	21638	129
5	9	9788075779113	Tiché roky	501	12913	36
6	11	9788075542533	Abeceda mzdové účetní 2020	18	12402	3
7	17	9788074919404	Hana	416	10166	28
8						
9						
10						
11						

Obrázek 15: *Výsledný excelový soubor.*

7 Co bych změnil

S výsledkem práce jsem spokojený, ale kdybych mohl vrátit čas tak bych udělal několik změn.

Podpůrný program bych nevytvářel jako program pro *Windows* ale jako webovou aplikaci. Usnadnilo by mi to hodně práce s aktualizací programu, protože bych ji mohl provézt na jednom místě a všichni uživatelé by k ní měli přístup. A hlavně bych si ušetřil starosti s instalací programu na 50 různých počítačích a s řešením problému přes vzdálenou plochu.

Mobilní aplikaci bych nevytvářel jako nativní aplikaci pro *Android* ale jako multiplatformní aplikaci pro *Android* a *iOS* zařízení pomocí *frameworku Flutter*. Zbytečně jsem tímto rozhodnutím zabránil třetině zaměstnanců k přístupu k aplikaci.

A nejvíc lituji toho, že jsem v aplikaci nepsal *unit* testy[21]. Vývoj aplikace by se zefektivnil kdybych se snažil dodržovat způsob vývoje TDD[22]. Ušetřil bych si mnoho času, který jsem ztratil ručním testovaní všech funkcí aplikace po každé větší změně a mohl bych si dovolit provádět větší refaktorování kódu a tím by se kód stal čitelnější.

[21] Unit testy - Psaní testů pro každou metodu

[22] Test-Driven Development - vývoj řízený testy

Závěr

Výsledkem této práce je mobilní aplikace, která i rok od svého vydání má přes 50 aktivních uživatelů, kteří si chválí především usnadnění přístupu k informacím a zjednodušení tvorby vratek a objednávek. A to mě velmi těší, protože jsem se bál, že jí nikdo používat nebude.

Při její tvorbě jsem se zdokonalil v programování v jazyku *Java*, zjistil jsem jak funguje operační systém *Android* a naučil jsem se architekturu MVVM.

Podpůrný program se moc nepoužívá, protože jeho používání bylo příliš zdlouhavé a nepohodlné. Na žádost firmy Knihy Dobrovský s.r.o. jsem do aplikace přidal možnost stahování dat z FTP serveru a odesílání vratek a objednávek přes email a tyto dvě funkce podpůrný program nahradily. I přesto jsem rád, že jsem ho vytvořil. Při jeho tvorbě jsem se naučil jak se tvoří uživatelské rozhraní v *JavaFX* a vyzkoušel jsem si jak v praxi funguje síťový protokol TCP.

Vyvíjení mobilní aplikace i podpůrného programu mě bavilo, nebylo to tak náročné jak jsem si na začátku myslel. Hlavní aplikace i pomocný program splňují požadovanou funkčnost a tedy věřím tomu, že jsem cíl práce splnil.

Conclusions

The result of this thesis is a mobile application, which even a year since its release has over 50 active users, who praise the facilitation of access to information and simplification of dealing with returns and orders. And that makes me very happy, because I was afraid that no one would use it.

In the course of its creation, I improved my programming skills in *Java*, I learned how the *Android* operating system works and I learned the architecture of MVVM.

The support program is not used much because it was too lengthy and inconvenient. At the request of Knihy Dobrovský s.r.o. I added to the application the possibility of downloading data from the FTP server and sending returns and orders via email, and these two functions replaced the support program.

Still, I'm glad I created it. During its creation, I learned how to create an user interface in *JavaFX* and tried how the TCP network protocol works in practice.

I enjoyed the development of the mobile application and the support program, it was not as demanding as I thought at the beginning. The main application and the utility program meet the required functionality and therefore I believe that I have met the goal of the work.

8 Obsah přiloženého CD/DVD

`README.PDF`
: Manuál pro testování. Je v tu i heslo k FTP pro otestování stahování dat analýzy z FTP.

`doc/`
: Text práce ve formátu pdf, vytvořený s použitím závazného stylu KI PrF UP v Olomouci pro závěrečné práce, včetně všech příloh, a všechny soubory potřebné pro bezproblémové vygenerování pdf- dokumentu textu (v ZIP archivu), tj. zdrojový text textu, vložené obrázky, apod.

`data/data.csv`
: Ukázková testovací data pro testování nahrávání dat bez FTP přístupu.

`app/hlavni/app-debug.apk`
: Instalační balíček pro hlavní aplikaci.

`app/podpurna/prevodnik.jar`
: Spouštěcí soubor k pomocnému programu.

`src/`
: Kompletní zdrojové texty hlavní aplikace a podpůrného programu.

`install/`
: V této složce je instalační balíček pro Javu 8.

Literatura

[1] ISO [online], IEC/IEEE International Standard-Systems and Software Engineering–vocabulary, 2017 [cit. 2019-10-23] dostupné z: https://standards.ieee.org/standard/24765-2017.html

[2] Statista [online], Installed base of smartphones by operating system from 2015 to 2017, 2019 [cit. 2019-10-23] dostupné z: https://www.statista.com/statistics/385001/smartphone-worldwide-installed-base-operating-systems/

[3] Permissions overview [online], Android Developers 2020 [cit. 2020-2-16] dostupné z: https://developer.android.com/guide/topics/permissions/overview