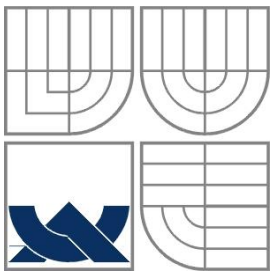
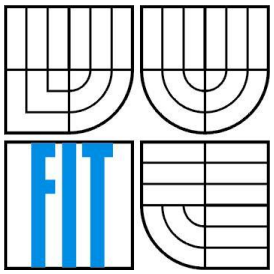
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2016 Bc. Petr Pyszko



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NATIVNÍ FRAMEWORK PRO UNIVERZÁLNÍ NABÍDKOVÝ SYSTÉM PRO PLATFORMU ANDROID

NATIVE ANDROID FRAMEWORK FOR A UNIVERSAL CATALOGUE SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Petr Pyszko

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Vladimír Bartík, Ph.D.

BRNO 2016

Abstrakt

Diplomová práce se zabývá návrhem a implementací frameworku pro operační systém Android. Tento framework je určen pro efektivní vývoj mobilních aplikací, jež jsou určeny k prezentaci produktů. Framework je navržen s ohledem na možnost přizpůsobení a změny implicitního chování.

Abstract

This thesis deals with design and implementation of framework for Android operating system. This framework is designed for effective development of mobile applications for products presentation. The framework considers the possibility of customization and change of its default behavior.

Klíčová slova

elektronické obchodování, mobilní obchodování, Android framework, mobilní aplikace

Keywords

electronic commerce, mobile commerce, m-commerce, Android framework, mobile application

Citace

PYSZKO Petr: Nativní framework pro univerzální nabídkový systém pro platformu Android, diplomová práce, Brno, FIT VUT v Brně, 2016

Nativní framework pro univerzální nabídkový systém pro platformu Android

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Vladimíra Bartíka Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Petr Pyszko
23. 5. 2016

Poděkování

Rád bych poděkoval svému vedoucímu práce Ing. Vladimíru Bartíkovi za odborné vedení, cenné rady a pomoc při tvorbě této práce.

Obsah

1 Úvod

V dnešní době se stále častěji přistupuje na internet z mobilních zařízení a mnoho webů již nabízí ke stažení specializovanou aplikaci. Mobilní aplikace nabízejí uživateli mnohem vyšší komfort než při prohlížení internetových stránek a pro internetové obchody jsou často mobilní aplikace výhodným zdrojem příjmu a nástrojem, jak si udržet zákazníka i pro budoucí nákupy.

V rámci této práce se zaměříme na návrh frameworku pro univerzální nabídkový systém na platformě Android. Rozebrány budou hlavní požadavky na takovýto systém a návrh, jak tyto požadavky naplnit. Framework bude také zohledňovat možnosti rozšíření a jeho praktické použití při implementaci na míru internetovému obchodu.

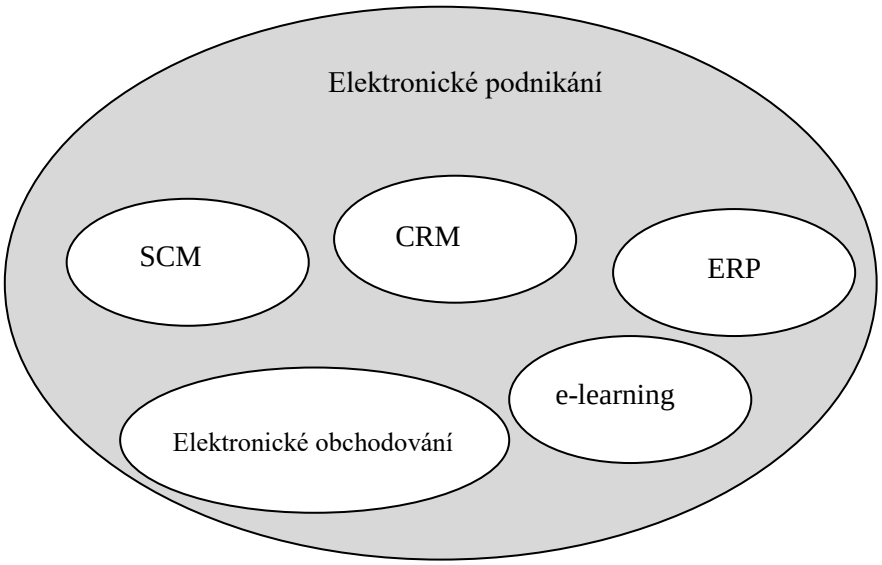
V úvodní kapitole se práce věnuje základním pojmům z oblasti elektronického obchodování a elektronického podnikání obecně. Vysvětleny jsou zde základní pojmy, modely obchodování a systémy pro elektronické podnikání s jejich základními funkcemi. V následující kapitole se zabýváme platformou Android, u které rozebereme její architekturu, vývojové nástroje, práci se zdrojovými soubory a tvorbu uživatelského rozhraní dle designových pravidel platformy. Ve čtvrté kapitole jsou rozebrána řešení pro univerzální výrobu mobilních aplikací pro mobilní obchodování. V následující kapitole je navržen framework s ohledem na jeho rozšíření vývojářem koncové aplikace. Šestá kapitola se věnuje implementaci frameworku. V poslední kapitole je popsána implementace ukázkové aplikace s použitím frameworku a možnosti jeho rozšíření.

2 Elektronické obchodování

2.1 Základní pojmy

Pojem **elektronické obchodování**, tj. **e-commerce** má více definic. Nejčastěji se však tímto pojmem rozumí prodej zboží nebo poskytování služeb přes internet. Jiné definice tento pojem popisují jako jakoukoli formu obchodování, při níž dochází k realizaci obchodních transakcí s využitím elektronických komunikačních prostředků. Nejčastěji využívanou službou bývá Word Wide Web.

Elektronické obchodování lze chápat jako jednu ze složek **elektronického podnikání,** které obsahuje kromě elektronického obchodování také procesy s ním spojené. Tyto procesy jsou realizovány pomocí moderních technologií využívajících internet pro zefektivnění interních i externích procesů v podniku [1].



Obrázek 1: Vztah elektronického podnikání a obchodování [1]

Nejčastěji se elektronické obchodování realizuje pomocí **internetového obchodu**. Jedná se o webovou aplikaci, která umožňuje nabízet zboží a služby a zprostředkovává obchodní transakce na internetu. Tato aplikace umožňuje také správu katalogu, objednávek a ostatních součástí nutných pro její provoz.

2.2 Obchodní modely elektronického obchodování

Elektronické obchodování se dělí zejména dle typu zúčastněných stran. Těmito stranami mohou být jak koncoví zákazníci, firmy, ale také státní správa. Mezi nejčastější druhy elektronického obchodování patří **Business to Business** (zkráceně **B2B**), kde dochází k obchodu mezi firmami

a **Business to Customer** (**B2C**), jež se zaměřuje na prodej koncovým zákazníkům. V současnosti již existuje daleko více modelů, které ovšem nebývají tak časté a dělení je v různých zdrojích rozdílné. Dalšími modely jsou [2]:

- **Customer to Customer** (**C2C**) – do níž patří oblast inzerčních a aukčních serverů;
- **Consumer to Business** (**C2B**) – spotřebitelé navrhují ceny, za které by měli zájem nakupovat zboží;
- **Business to Government** (**B2G**) – jedná se o vztah mezi podnikem a státní správou (např. elektronicky vedená výběrová řízení);
- **Buisiness to Employee** (**B2E**) **–** výměna informací mezi zaměstnanci a podnikem (např. zprostředkování vzdělání pomocí e-learningu).

V této práci se však zaměříme především na internetové obchodování B2C.

2.3 Systémy pro elektronické obchodování

Postavit systém pro elektronické obchodování na „zelené louce“ je poměrně finančně náročné. Vyvinutí takového systému vyžaduje důkladnou znalost problematiky a vývoj webové aplikace, který bývá časově náročný, protože musí obsahovat jak webovou prezentaci, tak i administraci samotného systému [3]. Výhodou tohoto přístupu však bývá to, že aplikace, která je postavená na míru danému typu obchodování, má lepší výkon, intuitivnější ovládání a může být také snadněji rozšiřitelná.

V dnešní době je mnoho internetových obchodů postavených na existujících systémech pro elektronické obchodování. Těch je již na trhu spousta a liší se v mnoha ohledech. E-commerce systémy mohou být krabicový software, který je po instalaci možné začít ihned používat, ale také sada nástrojů, která je určena pro zjednodušení a zrychlení vývoje konečného systému. Nabízené systémy se také liší mírou rozšiřitelnosti a přizpůsobitelnosti. Zatímco malý internetový obchod si často může vystačit s jednoduchým systémem, velké korporace často potřebují e-commerce systém vhodně integrovat s jejich stávajícími systémy. Na trhu existují e-commerce systémy, které jsou zdarma, ale i systémy, jejichž ceny licencí šplhají až do řádů milionů korun. E-commerce systémy bývají také často moduly systémů pro správu obsahu (Content Management Systems). V poslední době přibývají řešení, která fungují kompletně v cloudu. Výhodou je velice rychlé a jednoduché zprovoznění obchodu, naopak integrace a přizpůsobení těchto systémů je často prakticky nemožná, a proto se tento přístup hodí spíše pro malé obchody.

2.4 Požadavky na systémy pro elektronické obchodování

Jak již bylo řečeno v předchozí kapitole, e-commerce systémy se liší v mnoha ohledech, avšak je potřeba zmínit základní funkce, bez kterých by se žádný internetový obchod neobešel. Uvedené funkce nemusí nutně obsluhovat samotný systém, který je použit pro chod internetového obchodu, ale mohou být obsluhovány externími systémy, které s e-commece systémem komunikují.

2.4.1 Správa a prezentace katalogu produktů

Pro chod internetového obchodu je správa a prezentace produktů stěžejní a jedná se o nejvyužívanější část systému. Pro zákazníka je důležité, aby v katalogu našel produkt, který potřebuje. Katalog by měl být vhodně strukturován a měl by také obsahovat textové vyhledávání a filtrování produktů dle jejich parametrů.

Správa katalogu produktů je prováděna administrátorem a zahrnuje jak editaci, vytváření a mazání produktů, tak i editaci struktury katalogu a správu parametrů produktů. Kromě společných parametrů, jako název, popis či cena, mají různé typy produktů obecně rozdílné parametry, a je proto potřeba mít nástroj, jak tyto parametry pro různé typy produktů spravovat.

Jednotlivé produkty nemusí přímo odpovídat skladové položce, ale mohou mít také varianty, např. velikost a barva trička. Obecně se tyto varianty liší v určité podmnožině parametrů (např. cena, barva, velkost), a ne vždy jsou všechny kombinace dostupné k prodeji (např. šedé XXL tričko není na skladě). Některé systémy správu těchto „podproduktů“ nepodporují a je nutné pro každou variantu vytvářet samostatný produkt.

2.4.2 Nákupní proces

Před koupí nabízeného zboží musí zákazník projít nákupním procesem. Ten se skládá z různého počtu kroků, jimiž musí zákazník projít. Tento proces by měl být co nejvíce intuitivní a neměl by zákazníka příliš zdržovat. Složité či zdlouhavé nákupní procesy mohou často zákazníka odradit od koupě.

Prvním krokem nákupního procesu nejčastěji bývá náhled do virtuálního nákupního košíku. Ten obsahuje položky, které se zákazník chystá nakoupit. V následujících krocích zákazník zadává adresy, vyplňuje osobní údaje a vybírá způsob doručení a platby. V posledním kroku je nejčastěji zrekapitulována objednávka a zákazník je po potvrzení přesměrován na platební bránu, pokud si zvolil online platbu.

Tento tradiční nákupní proces se používá při prodeji hmotných produktů. Pokud např. internetový obchod prodává elektronické zboží, pak nemá smysl vybírat způsob doručení. Jiný nákupní proces je také použit např. při objednávce zájezdu, kdy zákazník často vyplňuje také osobní údaje jednotlivých účastníků. Při výběru e-commerce systému je proto vhodné zvážit, zda podporuje vybraný typ nákupního procesu, nebo do jaké míry je možné nákupní proces přizpůsobit, abychom dosáhli požadované funkcionality.

2.4.3 Správa objednávek

Přijatá objednávka je v systému zaevidována a je možné s ní dále pracovat. Objednávka se může obecně nacházet v různém stavu a zákazník by měl mít možnost stav své objednávky sledovat či dostávat notifikace o jeho změně. Tyto stavy bývají často konfigurovatelné a jejich nastavení koreluje vychystávací proces. U menších internetových obchodů jsou často objednávky spravovány přímo administrátorem obchodu, zatímco u velkých obchodů bývá proces integrován se skladovým systémem firmy a objednávka je automaticky aktualizována.

2.4.4 Tvorba a výpočet cen

V nejjednodušším případě se produktu přiřadí cena a ta je použita v nákupním procesu. V reálném světě však tento přístup nestačí a do výpočtu ceny produktu či celkové ceny objednávky vstupují nejrůznější faktory.

Nejčastějším problémem bývá správný výpočet daně, který je velice komplexní a závisí na mnoha faktorech, jimž se v této práci nebudeme dále věnovat. Pro tento výpočet bývají často využívány externí webové služby, které provádějí korektní kalkulaci.

Mimo daně mohou do ceny produktu také zasahovat i různé slevy. E-commerce systémy často umožňují definování různých typů slev – zákaznické, věrnostní, množstevní, sezónní atd. Slevy mohou být procentuální či pevné a dají se definovat řadou pravidel. Tyto slevy se pak aplikují na produkty, nebo jsou odečteny od celkové ceny objednávky.

V B2B internetovém obchodování jsou často stejné produkty nabízeny za různé ceny různým zákazníkům. Cena pak v tomto případě nebývá spjata přímo s produktem, ale vyhodnocuje se na základě určitých pravidel.

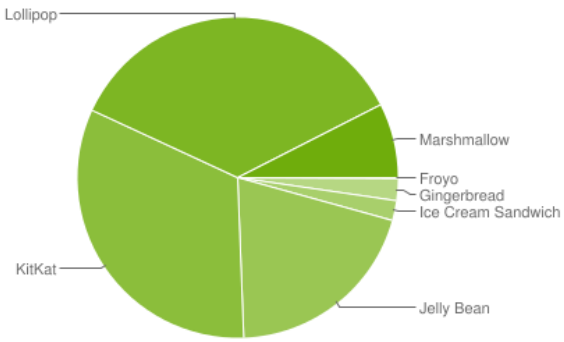
Některé e-commerce systémy umožňují pracovat i s více měnami a zákazník si pak může zvolit, v jaké měně bude nakupovat. Více měn však přináší také problémy s přepočtem ceny a aktualizací cen dle momentálního kurzu. Prvním přístupem je stanovovat různé ceny zvlášť pro různé měny. Tento přístup je jednodušší vzhledem k výpočtu ceny, ale přináší i problém s udržováním většího počtu cen. Druhým přístupem je mít cenu definovanou pouze v hlavní měně a cenu v jiných měnách počítat dle kurzu, který je však potřeba aktualizovat.

3 Platforma Android

V této kapitole se budeme zabývat operačním systémem Android. Probereme si základní témata operačního systému a vývoje mobilních aplikací. Zaměříme se především na oblasti, které jsou prerekvizitou pro vytvoření knihovny pro univerzální nabídkový systém. V tom se zejména zaměříme na vývoj znovupoužitelných komponent uživatelského rozhraní respektující materiální design. Druhým tématem bude komunikace se serverem pomocí formátu JSON s použitím kešování dat.

Operační systém je Android v současnosti nejrozšířenější platformou pro mobilní telefony a tablety. Operační systém je postaven na linuxovém jádře a je nyní vyvíjen společností Google Inc. pod open source licencí. Android byl primárně navržen pro použití pro mobilní zařízení s dotykovým displejem, avšak v dnešní době již jeho modifikované verze pohánějí i televizory (Android TV), palubní systémy aut (Android Car), chytré hodinky (Android Wear), brýle, mikropočítače, herní konzole a další druhy elektroniky.

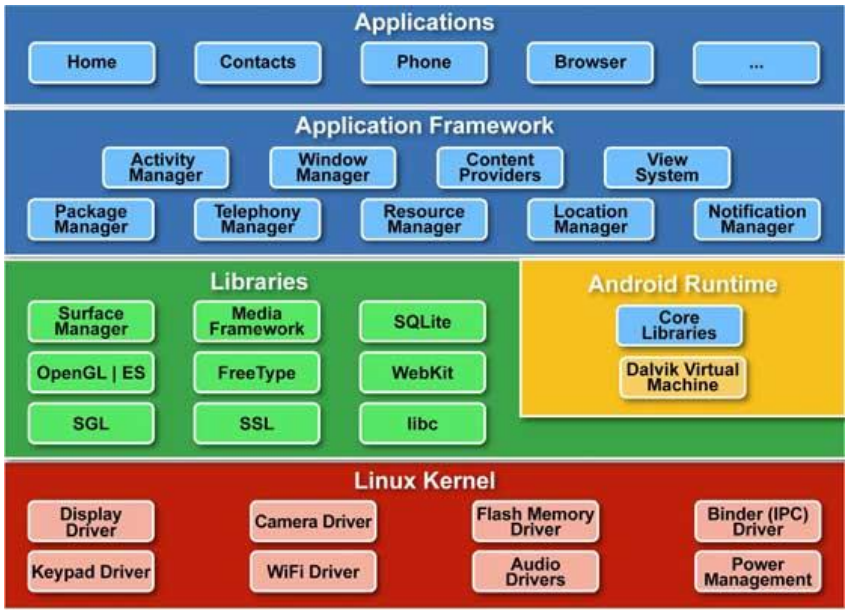
Operační systém android se nyní nachází ve verzi 6.0 Marshmallow. Distribuce nových verzí operačního systému na stávající zařízení bývá ovšem poměrně pomalá a výrobci většinou připravují nové verze Androidu jen pro vybrané modely po dobu okolo dvou let od jejich představení. Na obrázku Obrázek 2 vidíme podíl jednotlivých verzí aktivních mobilních zařízení ke dni 2. 5. 2016 [4]. Jak vidíme na obrázku, podíl aktuální verze Androidu je pouze 7,5 procenta.



Obrázek 2: Podíl verzí OS Android [4]

3.1 Architektura

Operační systém Android se skládá z pěti sekcí ve čtyřech vrstvách. Každá vrstva poskytuje vyšší vrstvě své služby. Nejnižší vrstvou je linuxové jádro a nejvyšší vrstva obsahuje již aplikace, které běží nad operačním systémem Android. Na obrázku Obrázek 3 vidíme jednotlivé vrstvy architektury OS Android [7].



Obrázek 3: Architektura OS Android [7]

3.1.1 Linuxové jádro

Tato nejnižší vrstva je tvořena linuxovým jádrem, jež se oproti klasickým linuxovým distribucím lehce liší, a je přizpůsobeno pro běh na mobilních zařízeních. Tato vrstva poskytuje základní systémovou funkcionalitu, jako např. správa procesů, správa paměti, či práce s vstupně výstupními operacemi. Další funkcí této vrstvy je poskytnutí abstrakce při používání hardwarových prvků, jako fotoaparát, klávesnice, displej atd. [7]

3.1.2 Knihovny

Nad linuxovým jádrem se nachází vrstva sady knihoven. Mezi nejdůležitější knihovny patří WebKit engine poskytující jádro webovým prohlížečům, klasické libc knihovny, knihovny pro práci s multimédii, SQL lite databázový systém, knihovny pro práci s grafikou a mnohé další. Aplikace tyto knihovny využívají skrz aplikační framework [7].

3.1.3 Běhové prostředí

Běhové prostředí androidu je tvořeno komponentou zvanou **Dalvik Virtual Machine**, což je obdoba **Java Virtual Machine**, ale speciálně navržena a optimalizována pro android. S příchodem androidu 5.0 je však tato komponenta nahrazena novým běhovým prostředím ART (Android Runtime). Toto běhové prostředí využívá tzv. ahead-of-time (česky s předstihem) kompilaci, díky níž se pomocí utility **dex2oat** aplikace přeloží z formátu **DEX** (Dalvik Executable) na spustitelný formát pro dané zařízení. Nové běhové prostředí vylepšuje výkon aplikací, má lepší správu paměti a umožňuje také lepší ladění a profilování aplikací [9].

3.1.4 Aplikační framefork

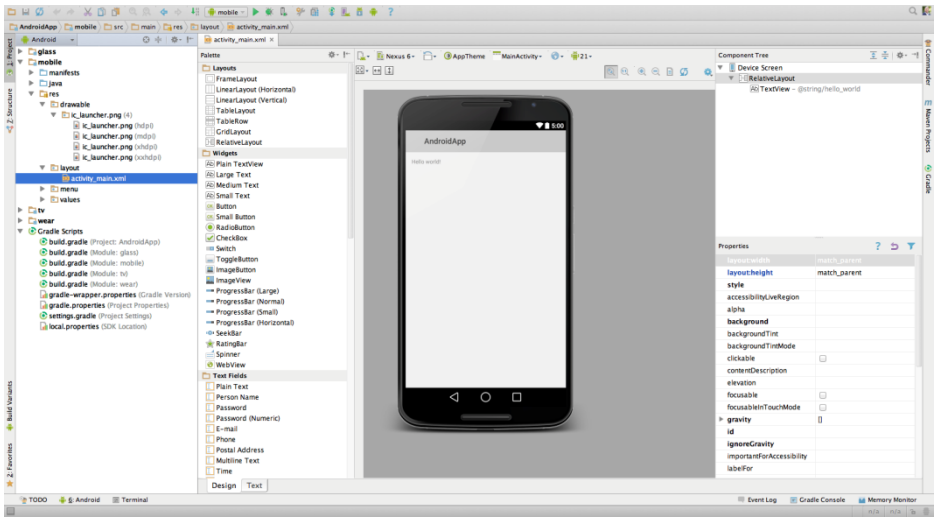
Tato vrstva poskytuje aplikacím vysokoúrovňové služby ve formě Java tříd. Aplikace využívají těchto služeb pro implementaci uživatelského rozhraní, práci se zdroji (texty, grafika, obrázky), s internetem, polohou, upozorněními, telefonními sítěmi atd [7].

3.1.5 Aplikace

Na vrcholu se nachází vrstva s aplikacemi. Ty jsou psány v jazyce Java a nacházejí se vždy jen v této nejvyšší vrstvě. Mezi tyto aplikace patří například internetový prohlížeč, kalendář, aplikace na správu kontaktů, hry a podobně. Aplikace pro OS Android se nejčastěji distribuují pomocí tržiště Google Play.

3.2 Vývojové nástroje

Pro vývoj aplikací se v momentální době doporučuje používat vývojové prostředí Android Studio, které bylo vyvinuto přímo pro účel vývoje na této platformě. Toto vývojové prostředí nabídne pohodlnou práci se zdrojovými soubory, WYSIWYG editor pro tvorbu uživatelského rozhraní, Gradle překladový systém nebo nástroje pro ladění a profilování aplikací. Tento nástroj také poskytuje podporu pro vývoj aplikací specializovaných na televize, nositelnou elektroniku či Google Glass brýle [6].



Obrázek 4: Tvorba uživateského rozhraní pomocí vývojového prostředí android studio [6]

Alternativou pro vývoj aplikací je zásuvný modul ADT (Android Developer Tools) do vývojového prostředí Eclipse. Výhodou tohoto vývojového prostředí je možnost použití NDK (Native Development Kit) pro implementace částí aplikace pomocí nativního kódu v jazyce C/C++. Tento přístup je vhodné využít pro implementaci výpočetně náročných operací, kterými jsou často herní frameworky, zpracování signálů nebo fyzikální simulace [5].

3.3 Vývoj komponent uživatelského rozhraní

Android obsahuje již v základu celou paletu komponent uživatelského rozhraní, které vývojář typicky používá během vývoje aplikace. Mezi hlavní typy patří komponenty definující rozvržení (layouts), komponenty pro zobrazování seznamů (GridView, ListView atd.) a formulářové prvky (inputs). Vývojář poté může použít WYSIWIG, nicméně mnohem častěji se používá přímo zápis pomocí formátu XML, kterým se vzhled uživatelského rozhraní definuje. Kromě atributů jednotlivých komponent může vývojář ovlivnit i jejich vzhled pomocí souborů stylů.

Pokud chce vývojář programově měnit nastavení komponent v kódu, typicky k navázání událostí uživatelské interakce, použije u komponenty atribut identifikátoru, díky němuž je poté schopen získat referenci na konkrétní instanci komponenty.

Kromě využívání Android komponent může vývojář vytvořit komponenty vlastní a ty použít ve více projektech. Tato možnost se hojně používá a pro platformu Android tak vzniklo velké množství často volně šiřitelných komponent a grafických knihoven pro tvorbu uživatelského rozhraní.

Typicky se v tomto případě používají následující přístupy k tvorbě vlastních komponent [12]:

- nová vlastní vykreslená komponenta s použitím 2D grafických operací;
- kombinace existujících komponent s určitým nastavením;
- rozšíření již existující komponenty s přidáním funkcionality, změnou chování či změnou vzhledu.

Obrázek Obrázek 5: Tvorba uživatelského rozhraní pomocí XML zápisu ilustruje definici vzhledu uživatelského rozhraní pomocí XML formátu. V komponentě rozvržení RelativeLayout je zasazen obrázek a popisek položky.

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center">

    <ImageView
        android:id="@+id/cart_image"
        android:src="@drawable/ic_cart"
        android:layout_width="50dp"
        android:layout_height="match_parent"
        android:paddingRight="8dp"
        android:paddingLeft="8dp"
        android:gravity="center"
        android:background="?attr/selectableItemBackgroundBorderless"
        android:scaleType="fitCenter"
        android:contentDescription="cart"
        />

    <TextView xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/cart_count"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="#ffffffff"
        android:gravity="center"
        android:text="35"
        android:layout_alignTop="@id/cart_image"
        android:layout_alignRight="@id/cart_image"
        android:layout_marginRight="8dp"
        android:layout_marginTop="8dp"
        android:paddingRight="4dp"
        android:paddingLeft="4dp"
        android:background="@drawable/shape_cart_count"/>
</RelativeLayout>
```

Obrázek 5: Tvorba uživatelského rozhraní pomocí XML zápisu

3.3.1 Customizace komponent

Pokud vývojář nemůže dosáhnout požadovaného chování či vzhledu s využitím vestavěných komponent, může si vytvořit vlastní pomocí dědičnosti. V tomto případě využije základní komponentu *View*, či kteroukoli specifičtější komponentu. Nejčastějším přístupem bývá:

1) Vytvoření nové třídy dědící z bázové *View* třídy (např. *TextView*).
2) Přidání konstruktoru s parametrem atributů, které jsou vytvořeny z XML souboru. Tímto způsobem může komponentu rozšířit o nové atributy, které určují počáteční nastavení komponenty.
3) Přidání metod obsluhující chování komponenty, reakce na uživatelské vstupy a obecnou logiku komponenty.
4) Přetížení metody *onDraw()* a *onMeasure()*. Díky metodě *onDraw()* je vývojáři umožněno provádět 2D grafické operace na instanci třídy *Canvas*. Pomocí metody *onMeasure()* vývojář určuje, jakou výšku a šířku bude mít koncová komponenta. Informace o velikosti komponenty jsou důležité zejména pro rodičovské elementy v rozvržení uživatelského rozhraní.

3.3.2 Složené komponenty

Pokud si vývojář postačí s použitím vestavěných komponent a chce nějakou skupinu zapouzdřit a využívat ji samostatně, může pro tento účel využít složení komponent. Typicky tak učiní pomocí rozšíření již existující komponenty rozvržení (např. *LinearLayout*). Nastavení a rozvržení může poté vývojář definovat pomocí XML, nebo komponenty nastavovat a přidávat programově v kódu.

Zbylou logiku a spoluprácí jednotlivých komponent vývojář zajistí postupem uvedeným v předchozí kapitole. Na rozdíl od předchozího postupu k vytváření vlastních komponent nemusí vývojář implementovat metody *onDraw()* a *onMeasure()*, jelikož jejich korektní implementaci obsahuje již bázová komponenta rozvržení.

3.4 Práce s aktivitami a fragmenty

V této kapitole se seznámíme se základními stavebními kameny vývoje aplikací pro platformu Android. Hlavním tématem budou aktivity, které představují jednotlivé uživatelské obrazovky, mezi nimiž se uživatel naviguje a interaguje s nimi. Projdeme si také komunikaci jednotlivých aktivit, jejich životní cyklus a jejich použití v kombinaci s fragmenty.

3.4.1 Aktivity

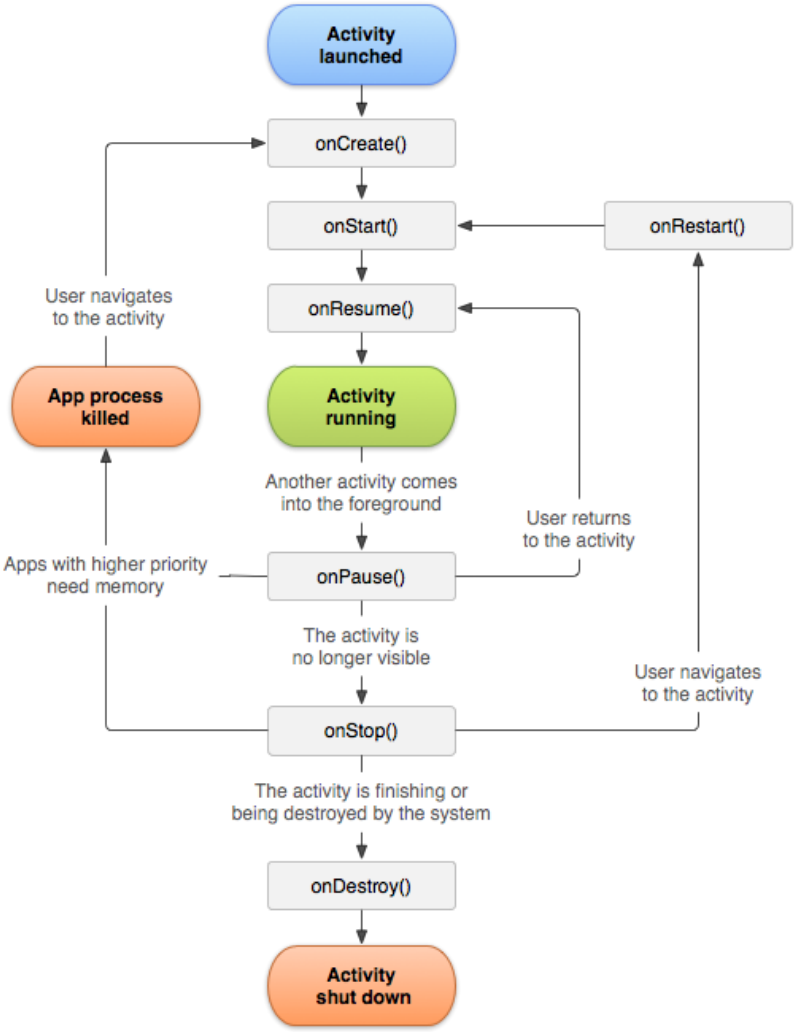
Aktivita je aplikační komponenta zobrazující uživateli obrazovku uživatelského rozhraní, se kterou může interagovat. Typicky je aktivita zobrazena přes celý displej mobilního zařízení, nicméně aktivita může být zobrazena i v menším výřezu nad ostatními obrazovkami. Aktivity mohou volající aktivitě vracet výsledky a lze spouštět i aktivity jiných aplikaci. Tímto způsobem jsou například implementovány platby za služby v aplikacích pomocí platební služby Google Play.

Aplikace má typicky jednu hlavní aktivitu, která je zobrazena po spuštění aplikace. Po spuštění jiné aktivity (např. pro zobrazení detailu objektu) je předchozí aktivita vložena do LIFO zásobníku aktivit, který nám zajistí návrat do původní aktivity v případě, že uživatel stiskne tlačítko pro zpětnou navigaci.

Po spuštění nové aktivity či jiné změně životního cyklu aktivity je původní aktivita notifikována pomocí akcí, na které může vývojář zareagovat – vytvoření, zastavení, obnovení či odstranění aktivity. Vývojář díky těmto akcím může například uvolnit zdroje při zastavení aktivity a naopak je znovu získat při obnovení.

Pro implementaci aktivity je nutné vytvořit třídu dědící z třídy *Activity*. V této třídě může vývojář implementovat metody akcí životního cyklu aktivity. Nejdůležitější akcí je metoda *onCreate(),* jež je vykonána při spuštění aktivity. V této metodě nejčastěji vývojář volá metodu *setContentView()* s parametrem názvu XML souboru definující rozložení komponent uživatelského rozhraní.

Níže uvedený diagram znázorňuje přechody mezi jednotlivými stavy aktivity. Stavy označují akce, které jsou vyvolány při přechodu aktivity do určitého stavu.

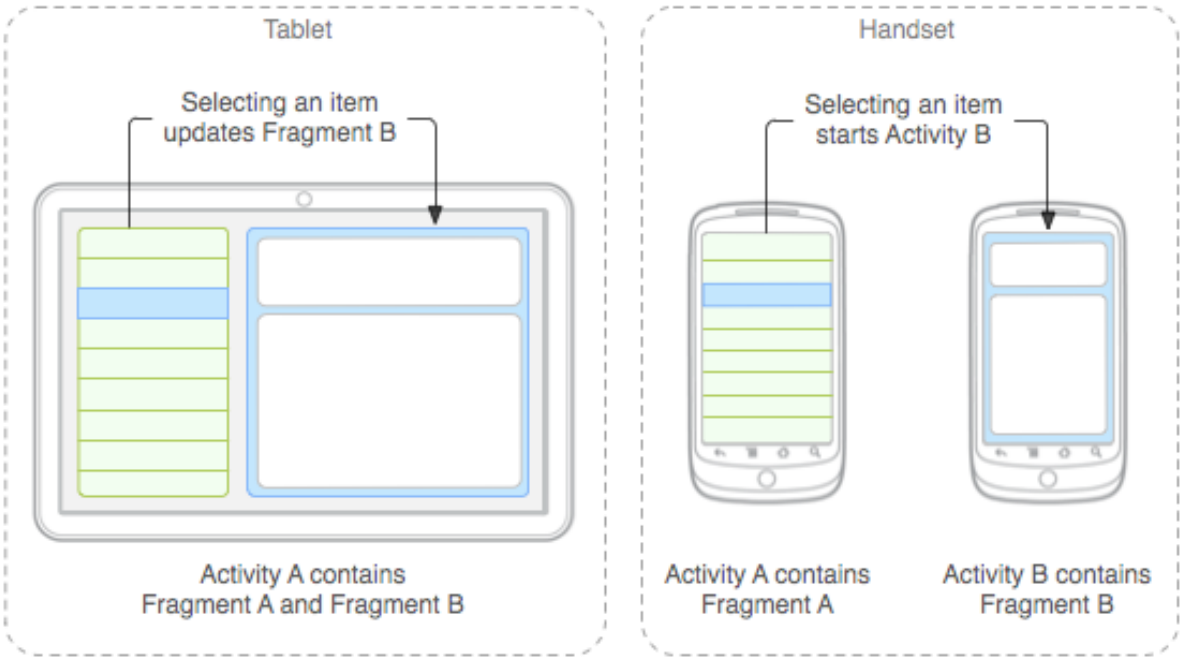


Obrázek 6: Životní cyklus aktivity [13]

3.4.2 Fragmenty

Fragmenty jsou oproti aktivitám pouze části uživatelského rozhraní. Fragmenty jsou v Androidu dostupné až od verze 3.0 a jejich hlavní výhodou oproti aktivitám je rychlost jejich načítání. Fragmenty lze použít ve více aktivitách či zobrazovat větší množství fragmentů na stránce. Rozdělení uživatelského rozhraní na fragmenty také podporuje modularitu aplikací.

Fragmenty lze vkládat do XML souborů rozvržení jako elementy, nebo pomocí komponenty *FrameLayout*, do které jsou pak fragmenty dynamicky vkládány či odebírány. Fragmenty mají stejně jako aktivity životní cyklus a při jejich dynamickém přepínání jsou také vkládány do zásobníku pro zpětnou navigaci uživatelem.



Obrázek 7: Rozložení více fragmentů na obrazovce [14]

3.5 Kešování dat

Kešování dat obecně má za úkol eliminovat čas potřebný k získání určitého objemu dat použitím rychlejšího úložiště. V kontextu vývoje aplikací pro platformu Android se nejčastěji jedná o eliminaci opakovaného přístupu k internetovým zdrojům nebo čtení souborů z interního úložiště (obrázky, datové či konfigurační soubory). Kapitola pojednává o technikách, které pro tento účel poskytuje přímo platforma Android, nicméně dostupné jsou i mnohé knihovny třetích stran.

3.5.1 Ukládání keš souborů

Na platformě Android má každá aplikace svoji privátní složku, do níž může aplikace vkládat, mazat či modifikovat své soubory. K této složce nemá uživatel (bez root práv) přístup. Mimo klasické úložiště souborů má zde vývojář přístup k uložení i keš souborů. Tato složka je při odinstalaci aplikace automaticky smazána. Pro přístup k těmto keš souborům je potřeba využít metodu *getCacheDir()*, jejíž návratová hodnota reprezentuje cestu k privátní složce pro uložení keš souborů. Pokud v mobilním zařízení dochází paměť, systém může tyto soubory smazat, aby uvolnil prostor. Na existenci takto vytvořených souborů se tedy vývojář nemůže spoléhat. Kapacita tohoto prostoru není nijak omezena, nicméně doporučuje se držet velikost této složky okolo 1MB [15].

3.5.2 Kešování bitmapových obrázků

Načtení a následné zobrazení jednoho obrázku z interní paměti zařízení není nijak extrémně výkonově náročné, nicméně problém nastává, když jich je na obrazovce více. Typicky se tento problém projevuje při použití komponent pro zobrazení seznamu položek s obrázky (např. galerie obrázků, seznam produktů atd.). Komponenty většinou uvolňují paměť alokovanou obrázkem, pokud se dostane mimo zobrazovanou část seznamu, nicméně pro plynulé posuvy je vhodné opakované načítání obrázku minimalizovat.

Pro tyto účely je vhodné obrázky kešovat v operační paměti nebo do úložiště telefonu (např. náhledy obrázků v galerii). Android pro toto použití nabízí třídu *LruCache* (last-recently-used cache) nebo *DiskLruCache*. Instancím těchto tříd lze poté nastavit maximální kapacitu keše a implementace se již postará o odstraňování nejdéle nepoužívaných položek. Zatímco keš v operační paměti může typicky obsahovat pouze malou část seznamu zobrazovaných obrázků, keš v úložišti zařízení se používá k urychlení načítání většího počtu obrázků. Keš v úložišti je daleko pomalejší, nicméně výkonově je stále rychlejší než opětovné zpracování bitmap (např. náhledy obrázků v galerii) [16].

3.5.3 Kešování stahovaných souborů

Základem pro zefektivnění načítání internetových zdrojů je načítání pouze dat, která potřebujeme. Typicky je vhodné například snížit rozlišení obrázku náhledu již na serveru a plné rozlišení stahovat až v případě nutnosti zobrazení detailu s plným rozlišením. V mobilních aplikacích často uživatel opakovaně zobrazuje stejný obsah. Takový obsah je proto vhodné kešovat. Pro kešování stahovaných souborů je možné použít informace z hlavičky HTTP protokolu. Tyto informace (čas poslední změny a čas exspirace) je možné využít pro eliminaci stahování a využití lokálně naklovaného souboru.

Platforma Android nabízí mechanizmus, jak tuto rutinu zjednodušit. V níže uvedeném úryvku kódu je uvedeno, jak zapnout lokální keš pro stahované soubory. V kódu je použita reflexe a jako parametry volané metody jsou použity velikost lokální http keše a cesta ke keš adresáři [17].

```
private void enableHttpResponseCache() {
  try {
    long httpCacheSize = 10 * 1024 * 1024; // 10 MiB
    File httpCacheDir = new File(getCacheDir(), "http");
    Class.forName("android.net.http.HttpResponseCache")
         .getMethod("install", File.class, long.class)
         .invoke(null, httpCacheDir, httpCacheSize);
  } catch (Exception httpResponseCacheNotAvailable) {
    Log.d(TAG, "HTTP response cache is unavailable.");
  }
}
```

Obrázek 8: Aktivace lokální HTTP [17]

3.6 Zpracování formátu JSON

Formát JSON (JavaScript object notation) je jeden z nejpoužívanějších formátů pro přenos dat mezi serverem a klientskou mobilní aplikací. Jedná se o textový formát a formát dat vychází ze zápisu objektů a polí jazyka JavaScript. Formát je dobře čitelný člověkem a je vhodný i pro strojové zpracování. Oproti formátu XML bývá často úspornější.

Pro zpracování tohoto formátu se na platformě nejčastěji používají třídy *JSONObject* a *JsonReader*. *JSONObject* načítá celý vstupní JSON objekt do paměti. Následné zpracování probíhá doptáváním hodnot na základě klíče. Návratovou hodnotou může být objekt, pole nebo konkrétní hodnota. V případě čtení objemnějších JSON objektů s důrazem na rychlost se doporučuje použití *JsonReader*. Ten umožňuje čtení pomocí proudu, čímž šetří paměťovou i časovou náročnost.

Zpracování výše uvedenými způsoby je lehce přizpůsobitelné, nicméně nutí vývojáře psát nový kód pro každý typ zpracovávaného objektu. Některé veřejně dostupné knihovny umožňují automatickou serializaci a deserializaci Java objektů do formátu JSON bez nutnosti tvorby kódu

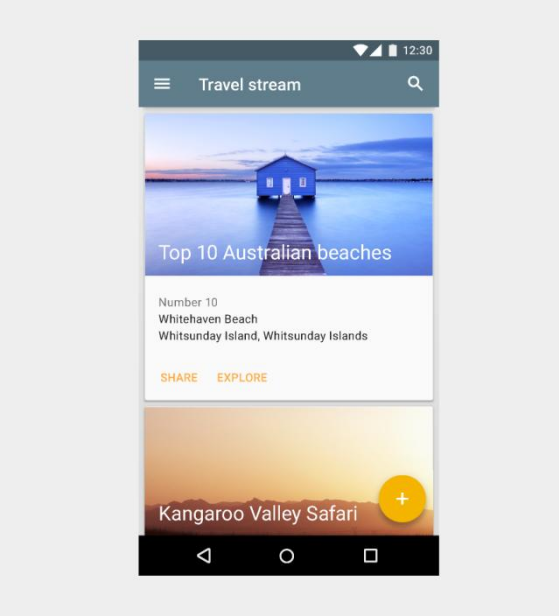
určeného pro jeden typ objektu. Knihovny využívají typicky systému Java anotací či reflexe, kdy program za běhu zjistí formát Java objektu a automaticky ho převede do formátu JSON. Nejznámější a nejpoužívanější knihovnou pro Android je knihovna **google-gson**.

3.7 Designový jazyk „material design“

V této kapitole si probereme základy vizuálního jazyka doporučovaného společností Google pro vývoj aplikací na platformě Android. Jazyk byl představen spolu s Android verzí 5.0 v roce 2014. Design je inspirován klasickým povrchem materiálu a jeho vrstvami, které mají stíny. Material design lze použít na mobilních zařízeních běžících na Androidu verze 2.1 a vyšším. Společnost Google zařídila kompatibilitu díky knihovně appcompat v7.

3.7.1 Materiálové vrstvy

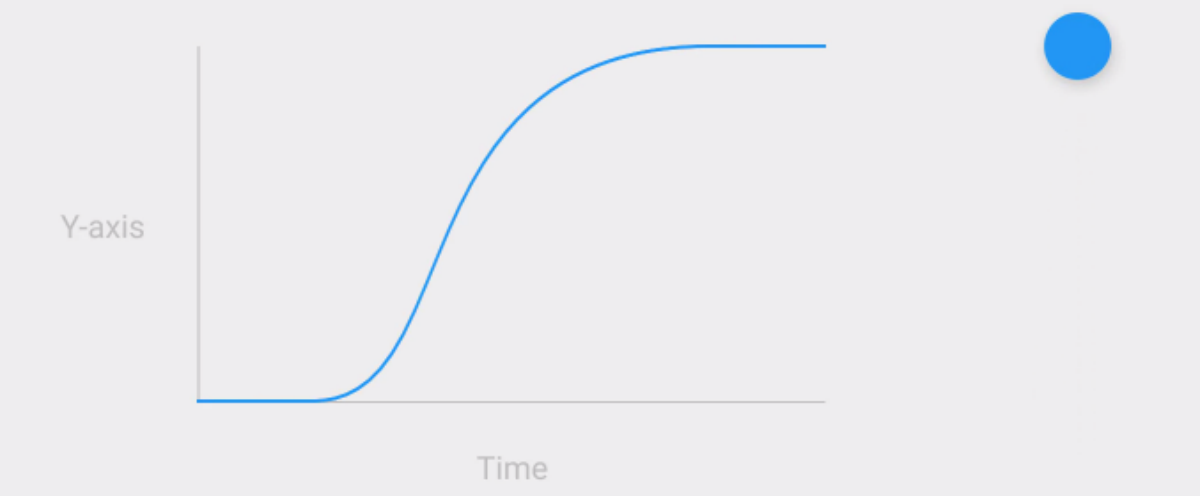
Designový jazyk vnímá uživatelské rozhraní jako třídimenzionální prostor, který obsahuje osvětlení, vrstvy materiálu a stíny jimi vrhané. Materiální vrstva je vnímána jako určitá část uživatelského rozhraní, která zobrazuje určitý obsah. Vrstvy mají vždy stejnou tloušťku 1dp (jednotka nezávislá na jemnosti rozlišení zařízení). Pro každý element uživatelského rozhraní lze definovat atribut *elevation*, který určuje výšku položení elementu nad spodní vrstvou. V doporučeních pro vývoj aplikací dle material designu je určeno, jak nastavovat atribut *elevation* pro konkrétní prvky uživatelského rozhraní.



Obrázek 9: Uživatelské rozhraní s použitím vrstev a stínů [18]

3.7.2 Pohyby a animace uživatelského rozhraní

Animace mají v material designu důležitou roli. Pomáhají uživateli s navigací, poskytují odezvu na uživatelské akce a vytvářejí dojem plynulé a dynamické aplikace. Animace by měly reflektovat běžné chování objektů z reálného světa. Typicky tak animované elementy nemění svou pozici s lineární rychlostí, nýbrž postupně rychlost naberou a také ztratí. Animace také často respektují gravitaci.



Obrázek 10: Časový průběh animace elementu grafického rozhraní po ose Y [18]

Použití animací se také doporučuje mezi jednotlivými obrazovkami uživatelského rozhraní. Například při přechodu ze seznamu položek na obrazovku detailu položky. Při přechodu by měly být společné prvky (typicky obrázek a nadpis) plynule animovány, čehož je možné dosáhnout sdílením prvků mezi fragmenty či aktivitami.

3.7.3 Doporučení při vývoji aplikací dle materiálního designu

V předchozích kapitolách jsme si popsali základní pilíře materiálního designu. Kromě těchto zásad obsahuje oficiální příručka také velké množství doporučení při tvorbě designu aplikací. Následující seznam obsahuje hlavní témata, kterým se materiální design věnuje [18]:

- **Barvy** – použití barevné palety s důrazem na čitelnost a použití u jednotlivých komponent uživatelského rozhraní;
- **Ikony** – grafické zpracování ikon určených pro prezentaci aplikace nebo použití jako akčních prvků v uživatelském rozhraní;
- **Typografie** – typografické zásady u jednotlivých komponent
- **Rozložení** – popis struktury, rozložení komponent a jejich správné pozice;
- **Komponenty** – doporučení při kompozici uživatelského rozhraní, správná volba uživatelských komponent v různých případech použití;
- **Návrhové vzory** – typické řešení navigace, notifikací, gest a dalších oblastí vývoje grafického rozhraní.

3.8 Animace uživatelského rozhraní

Jak již bylo uvedeno v kapitole o materiálním designu, animace jsou důležitou součástí uživatelského rozhraní. V této kapitole si popíšeme základní přístupy ke tvorbě těchto animací na platformě Android.

V kontextu vývoje mobilních aplikací budeme vnímat animace jako změny vizuálních vlastností jednotlivých prvků uživatelského rozhraní. Animovat tak můžeme jednotlivé komponenty,

komponenty rozvržení, přechody fragmentů i aktivit. V Androidu se nacházejí dva frameworky pro tvorbu animací – animace vlastností (Property animations) a prvků (View animations). Doporučované je použití novějšího frameworku animace vlastností, nicméně pro přechody fragmentů a aktivit se stále používají animace prvků. Animace lze vytvářet programově, nebo je mít definované v XML souboru. Animace lze kombinovat, spouštět opakovaně a reagovat na události animací (krok animace, skončení animace atd.). Animacím se tedy nastaví změny vlastností, doba trvání a také způsob interpolace. Ten určuje průběh animace (např. lineární). Android poskytuje poměrně obsáhlou sadu předdefinovaných interpolací, nicméně je možné si napsat vlastní [19].

Úryvek kódu uvedený níže ilustruje definování animace pomocí formátu XML a jeho následné použití v jazyce Java.

```
<set android:ordering="sequentially">
    <set>
        <objectAnimator
            android:propertyName="x"
            android:duration="500"
            android:valueTo="400"
            android:valueType="intType"/>
        <objectAnimator
            android:propertyName="y"
            android:duration="500"
            android:valueTo="300"
            android:valueType="intType"/>
    </set>
    <objectAnimator
        android:propertyName="alpha"
        android:duration="500"
        android:valueTo="1f"/>
</set>
```

```
AnimatorSet set = (AnimatorSet) AnimatorInflater.loadAnimator(myContext,
    R.anim.property_animator);
set.setTarget(myObject);
set.start();
```

Obrázek 11: XML definice a použití animace [20]

4 Elektronické obchodování na platformě android

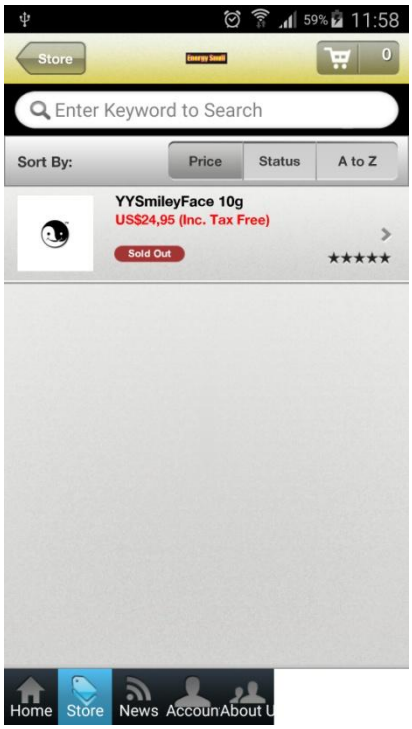
S popularizací mobilních zařízení vzrostl také zájem o nakupování za pomocí mobilního zařízení. Toto odvětví internetového obchodování se označuje jako **m-commerce** (mobilní obchodování). Nakupování přes mobilní zařízení s sebou přináší mnoho výhod. Nativní aplikace běží rychleji a nabídnou nakupujícímu daleko větší uživatelskou přívětivost než nakupovaní pomocí internetového prohlížeče mobilního zařízení. Pro prodávající je to další nástroj, jak si udržet zákazníka i pro jeho budoucí nákupy. Další výhodou pro prodávající je možnost zákazníka přímo oslovit pomocí „push notifikací“, což bývá často účinnější než email marketing. Další výhodou může být i práce v offline režimu, kdy je katalog stažen přímo do paměti mobilního zařízení a zákazník si může nabídku prohlédnout i bez připojení k internetu.

Většina mobilních aplikací je v dnešní době vyvíjena přímo na míru danému internetovému obchodu, na trhu se však už začínají objevovat i řešení, která nabízejí automatickou výrobu nativních aplikací po nahrání loga firmy, stylů a odkazu na konektor k e-commerce systému. Často se však jedná jen o nativní obal, ve kterém běží mobilní verze internetového obchodu. V následujících kapitolách si rozebereme vybrané existující m-commerce řešení.

4.1 MobiCart

Toto placené řešení nabízí jednoduchou možnost vytvoření mobilních aplikací pro platformy iOS, Android nebo internetový obchod postavený na technologii HTML5. Správa produktů a objednávek je možná přes jednoduché administrační rozhraní, přičemž celý systém je hostován na serverech Amazon Web Service. Systém je možné propojit s vaším stávajícím systémem pomocí aplikačního rozhraní, nebo využít nabízených konektorů k e-commerce systémům jako Magento či Prestashop [10].

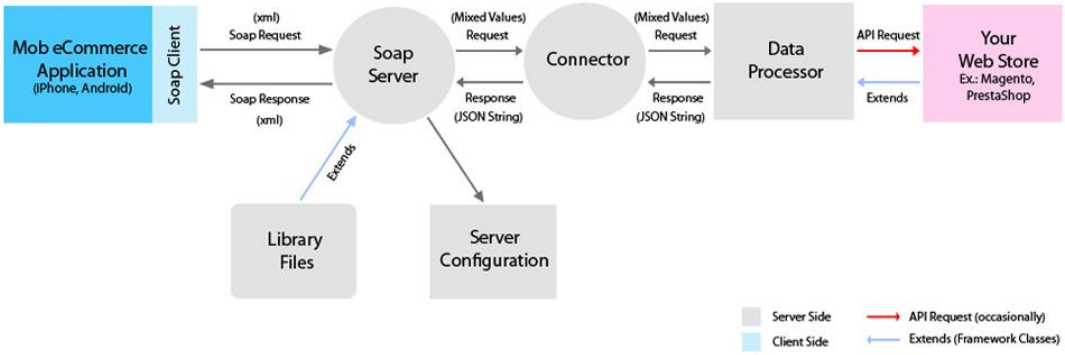
Výroba nativní aplikace probíhá pomocí jednoduché webové utility a po stažení hotové aplikace je možné ji ihned zveřejnit. Aplikace podporují push notifikace a obsahují veškerou funkcionalitu nutnou pro uskutečnění nákupu. Nevýhodou aplikací je však poněkud zastaralý vzhled a nemožnost přizpůsobení konkrétní aplikace. Zatímco na iOS uživatelské rozhraní odpovídá aplikacím pro verzi iOS 5, na Androidu vypadají aplikace téměř stejně jako na iOS, což je vzhledem k naprosto odlišnému způsobu navigace v aplikacích pro Android problém. Dalšími nevýhodami pro mobilní zařízení s operačním systémem Android je chybějící podpora používání aplikace v režimu na šířku. Aplikace také není přizpůsobena pro požití na tabletu.



Obrázek 12: Nativní MobiCart Android aplikace na zařízení Samsung Galaxy Note 3

4.2 Mob eCommerce

Tento framework se velice podobá řešení MobiCart. Výhodou tohoto řešení je propracovanější integrace s e-commerce systémy. K frameworku existují konektory pro Magento, Prestashop, Shopify a dalším e-commerce systémů. Framework podporuje vícejazyčné aplikace a k dostání jsou i zásuvné moduly pro platební brány. Komunikace mobilní aplikace neprobíhá přímo s e-commerce systémem, ale je zde využit SOAP server, který funguje jako prostředník [11]. Aplikace vytvořené tímto frameworkem vypadají lépe než u řešení MobiCart, avšak verze pro Android nerespektuje klasický vzhled aplikací pro tuto platformu.



Obrázek 13: Schéma architektury Mob Ecommerce frameworku [11]

5 Návrh frameworku pro nabídkový systém

V této kapitole se budeme věnovat návrhu frameworku, který zjednoduší práci vývojářům nativních m-commerce aplikací pro platformu Android. Tento framework bude sloužit zejména k usnadnění implementace prezentace produktů, hledání a filtrování v katalogu. Další funkcí bude prezentace a propagace slevových akcí či jiných obchodních sdělení.

5.1 Požadavky na framework

Hlavním účelem frameworku je zjednodušení tvorby katalogu. Ten by měl obstarávat základní funkcionalitu, jež bude odpovídat struktuře katalogu webového internetového obchodu. Vývojář, který bude tento framework využívat, by měl mít dostatečnou volnost pro přizpůsobení vzhledu a chování jednotlivých částí.

5.1.1 Úvodní obrazovka aplikace

Na úvodní obrazovce aplikace prodejci nejčastěji zobrazují seznam aktuálních slevových akcí, reklamní bannery či vybrané produkty. Tyto objekty bude vracet e-commerce server. Obecně se bude jednat o jakýkoli text s obrázkem na pozadí. Framework by se měl postarat o zobrazování těchto akcí, načítání dodatečných akcí při „scrollování“ a kešování obrázků pro vyšší výkon. Vzhled jednotlivých akcí by měl být přizpůsobitelný. Událost při kliknutí na akci bude možné nahradit libovolným kódem. Implicitní chování po kliknutí na akci bude definované dle typu akce přímo frameworkem (např. slevová akce na produkt přesměruje uživatele na obrazovku s detailem produktu).

5.1.2 Struktura katalogu

Katalog produktů má obvykle stromovou strukturu. Kořenovým uzlem bývá samotný katalog a listovými uzly jednotlivé skladové položky. Mimo tyto typy uzlů existují v katalogu kategorie produktů, jež mohou obsahovat podkategorie nebo produkty. Produktové uzly mohou být listové, nebo obsahovat produktové varianty, které odpovídají skladovým položkám. Konkrétní produkt může odpovídat i více uzlům této struktury.

5.1.3 Kategorie produktů

Jak již bylo zmíněno výše, kategorie produktů obsahuje podkategorie nebo produkty. Pro jednotný vzhled a chování katalogu nebude framework podporovat kategorie se smíšenými typy potomků.

Pokud kategorie obsahuje podkategorie, pak bude celý seznam stažen ze serveru a zobrazen uživateli. Uživatel nebude mít možnost tento seznam podkategorií nijak filtrovat či řadit. Podkategorie v seznamu bude položka obsahující obrázek a název. Programátor využívající framework bude mít možnost vzhled podkategorie přizpůsobit potřebám konkrétního obchodu. Položky jednotlivých podkategorií budou zobrazovány klasicky pod sebou, nebo budou tvořit záložky (komponenta tabview), což bude záležet na typu kategorie, kterou vrátí e-commerce server.

Pokud bude kategorie obsahovat produkty, uživateli se zobrazí jejich seznam. Zobrazení jednotlivých produktů bude opět možné programátorem upravit. Framework bude také podporovat

zobrazení více náhledů produktů na jednom řádku a počet sloupců této mřížky se bude přizpůsobovat rozměrům displeje mobilního zařízení. Tyto produkty budou podporovat také filtrování a řazení. Filtrování a řazení bude prováděno pomocí frameworkem definovaných uživatelských prvků. Každá kategorie může obecně obsahovat jiný typ filtrování či řazení, a proto budou typy filtrů definovány kategorií produktů vrácenou serverem.

5.1.4 Detail produktu

Po kliknutí na položku náhledu se uživateli zobrazí detail produktu, zde se bude nacházet galerie fotografií produktu, popis produktu, seznam parametrů, cena a komponenta pro výběr varianty produktu. Tento seznam komponent bude rozšiřitelný vývojářem aplikace a také hlavní komponenty budou nahraditelné jakýmkoli kódem.

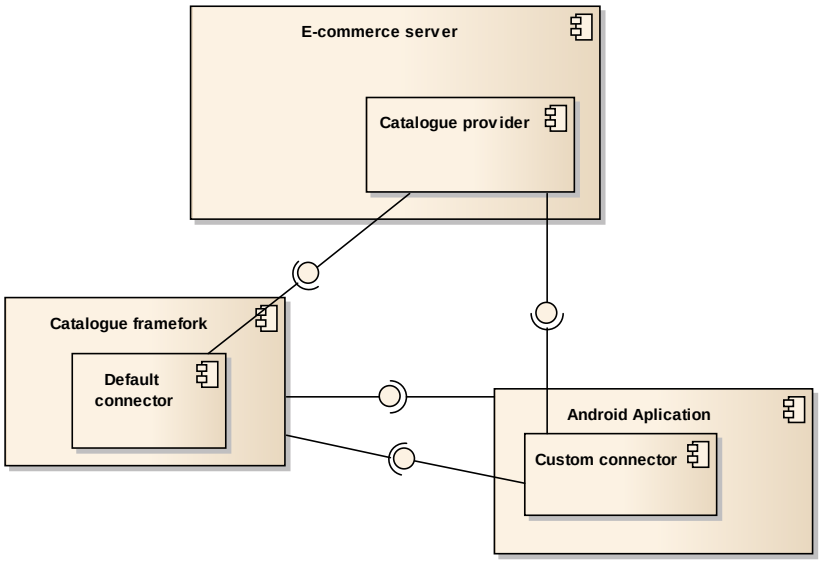
Výběr varianty produktu bude probíhat pomocí frameworkem definovaných komponent. Jaké komponenty se pro výběr varianty vyberou, bude určovat samotný objekt produktu navrácený serverem. Vybraná varianta může ovlivnit jakýkoli parametr produktu a je potřeba zařídit mechanismus, aby se po výběru varianty zobrazily komponenty s korektními údaji.

Popis produktu může obsahovat i složité formátování, a proto bude framework předepisovat formát HTML. Komponenta pro popis produktu bude umět popis s tímto formátem správně zobrazit.

5.2 Návrh použití frameworku

Na níže uvedeném obrázku je znázorněno využití frameworku koncovou aplikací. Komunikace s e-commerce serverem může probíhat dvěma možnými způsoby. Prvním způsobem je použití frameworkem definovaného formátu přenášených dat a implicitní konektor (**default connector**) bude schopen s e-commece serverem komunikovat a získávat potřebné objekty. Při použití tohoto způsobu komunikace bude muset server obsahovat rozhraní, jež bude poskytovat data ve formátu předepsaném frameworkem.

Pokud e-commerce systém již obsahuje své vlastní rozhraní, vývojář si napíše vlastní konektor (**custom connector**), který bude implementovat komunikaci se serverem a poskytovat data frameworku.



Obrázek 14: Propojení frameworku s e-commerce serverem a koncovou aplikací

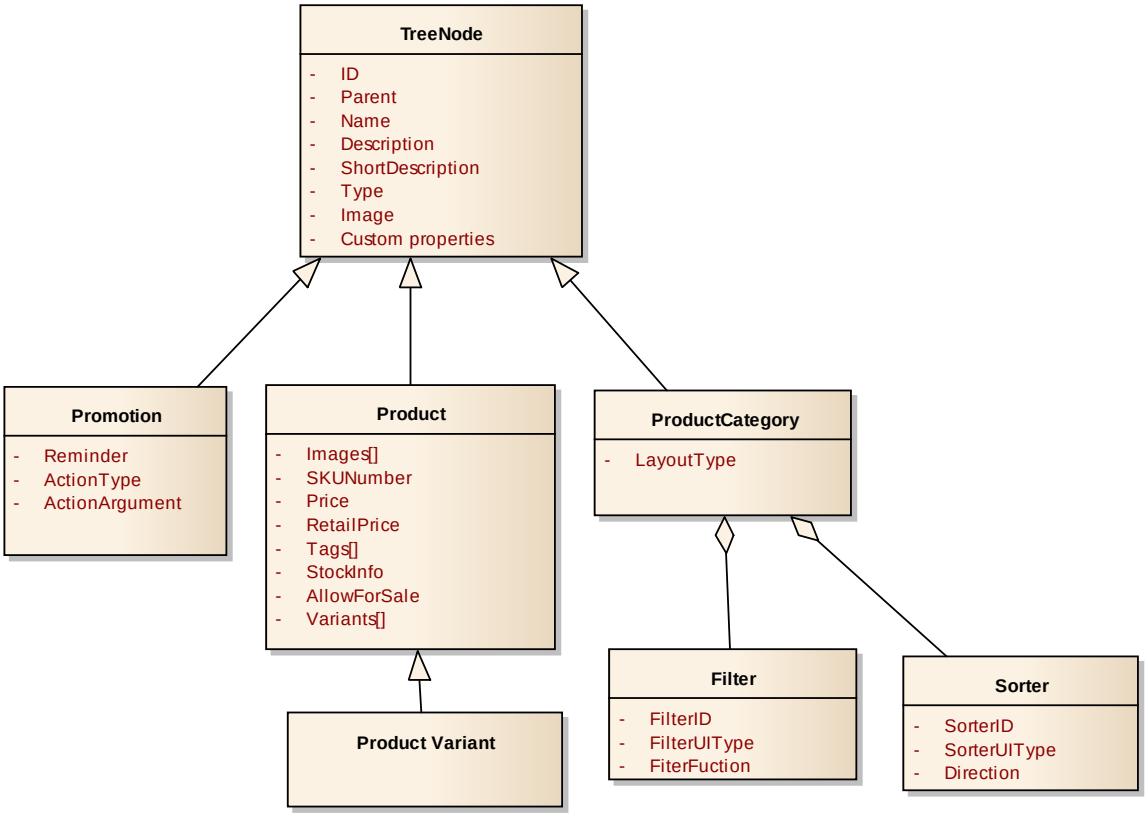
5.3 Datové struktury frameworku

Jak již bylo zmíněno, katalog je stromová struktura. Základním stavebním kamenem bude proto třída *TreeNode*, která bude obsahovat základní parametry, jež jsou společné pro produkty, kategorie či reklamní sdělení (*Promotion*).

Třída reprezentující produkt bude *TreeNode* rozšiřovat o parametry jako cena, dostupnost, seznam obrázků atd. Produkt může obsahovat seznam variant, které jsou reprezentovány třídou *ProductVariant*, jež se může od produktu lišit libovolnými parametry.

Třída *ProductCategory* zastupuje kategorie produktů. Ty se mnohou lišit použitým typem rozložení produktů. Kategorie produktů bude také obsahovat libovolné množství filtrů (*Filter*) a tříd reprezentující možnosti řazení (*Sorter*).

Odpověď e-commerce serveru může obsahovat i atributy, které nejsou definovány v jednotlivých třídách. V tomto případě budou nadbývající atributy uloženy v kolekci *CustomProperties*, která je reprezentována hašovací tabulkou, u níž bude klíčem název atributu a hodnotou textový řetězec. Pro vývojáře bude toto možnost, jak si k objektům přiložit dodatečné informace pro rozšíření funkcionality aplikace.



Obrázek 15: Diagram tříd katalogu

5.4 Komunikace s e-commerce systémem

Implicitní konektor bude pro přenos dat používat protokol HTTP a přenášené objekty budou ve formátu JSON. Formát JSON použijeme pro reprezentaci objektů přenášené ve směru ze serveru na klienta. Dotazy směřované na server budou reprezentovány jednoduchým GET dotazem s parametry, kterými mohou být identifikátor uzlu katalogu, hodnoty filtrů či směr řazení dle vybraného parametru. Následující ukázky dotazů a odpovědí znázorňují způsob komunikace e-commerce serveru a konektoru frameworku.

Příklad 1: Ukázka produktové kategorie s náhledy produktů, filtry a řazením


```
{
    "Name": "Android smartphones",
    "ID": 46901,
    "Image": "http://android-smartphones.jpg",
    "Type": "PRODUCT_LISTING",
    "parentID": 2587,
    "filters": [
    {
        "UIType": "DROPDOWN",
        "PropertyName": "Manufacturer",
        "Options": {
            "LG": 22,
            "Samsung": 43
        }
    },
    {
        "UIType": "CHECKBOXES",
        "PropertyName": "RAM",
        "Options": {
            "1GB": 0,
            "2GB": 1
        }
    }
  ],
    "sorters": [
    {
        "UIType": "BUTTON",
        "Direction": "ASC|DES",
        "PropertyName": "Price"
    }
  ],
    "Products": [
    {
        "ID": 5023,
        "Image": "http://example.com/G2.jpg",
        "Name": "LG G2",
        "Price": "$300",
        "retailPrice": "$380",
        "Manufacturer": "LG"
    },
    {
        "ID": 5063,
        "Image": "http://example.com/N4.jpg",
        "Name": "Samsung galaxy note 4",
        "Price": "$700",
        "Manufacturer": "Samsung"
    },
    {
        "ID": 5025,
        "Image": "http://example.com/N5.jpg",
        "Name": "Nexus 5",
        "Price": "$300",
        "Manufacturer": "LG"
    }
  ]
}
```


url: *ecommerce-server.com/catalogueprovider?nodeid=46901*

Příklad 2: Detail produktu se třemi variantami

```
{
    "ID": 5023,
    "Name": "LG G2",
    "Description": "Featuring an intuitive rear-key placement and a 2.26 GHz Qualcomm Snapdragon 800",
    "Price": "$300",
    "retailPrice": "$380",
    "Manufacturer": "LG",
    "RAM": "2GB",
    "Procesor": "Snapdragon 800 (4 cores)",
    "Size": "142x70x9",
    "Camera": "13mpx",
    "DisplaySize": "5.2\""
    "Images": [
    "http://example.com/G2_1.jpg",
    "http://example.com/G2_2.jpg",
    "http://example.com/G2_3.jpg"
    ],
    "guide-book" : " http://example.com/G2_guide.pdf"
    "Variants": [
    {
        "Price": "$300",
        "Name": "Black 16GB",
    },
    {
        "Price": "$350",
        "Name": "White 32GB",
    }
  ]
}
```

url: *ecommerce-server.com/catalogueprovider?nodeid=5023*

Příklad 3: Produktová kategorie s dvěma podkategoriemi

```
{
    "Name": "Smartphones",
    "ID": 2587,
    "Image": "http://smartphones.jpg",
    "type": "SECTION",
    "parentID": 234,
    "Categories": [
        {
            "Name": "Android smartphones",
            "ID": 46901,
            "Image": "http://android-smartphones.jpg",
            "type": "SECTION",
            "parentID": 2587,
            "ProductsCount": 129
        },
        {
            "Name": "iPhones",
            "ID": 48666,
            "Image": "http://ios-smartphones.jpg",
            "type": "SECTION",
            "parentID": 2587,
            "ProductsCount": 13
        }
    ]
}
```

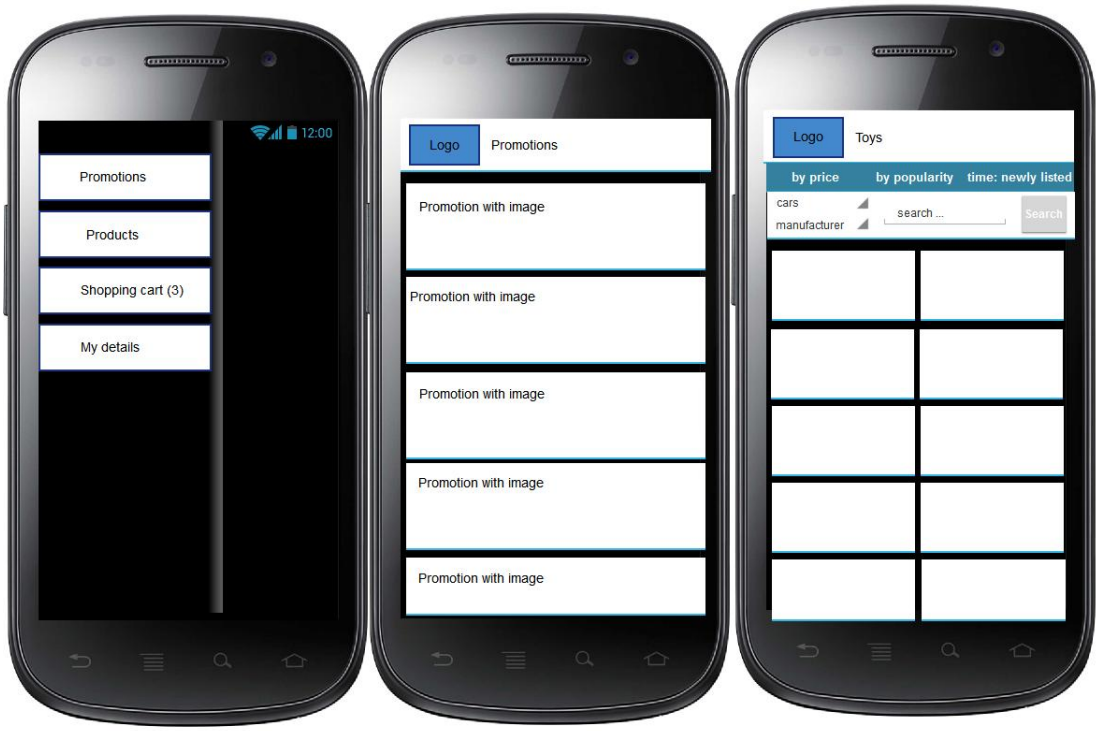
url: *ecommerce-server.com/catalogueprovider?nodeid=2587*

Jak vidíme na ukázkách, obrázky či doplňkové soubory nejsou přiloženy přímo v objektu, ale objekt obsahuje pouze odkaz. Obrázky se stahují asynchronně a jsou kešovány. Atributy, jež nebudou představovat obrázek produktu či kategorie, budou zobrazovány jako odkaz. Toto chování může vývojář ovlivnit implementací vlastní komponenty.

5.5 Návrh uživatelského rozhraní

Struktura a způsob navigace bude určená frameworkem a vývojář si ji nebude moci příliš přizpůsobit. Stylování, použitá loga, obrázky a ikony si bude moci vývojář měnit pomocí XML zdrojových souborů, které budou dědit z tématu definovaného frameworkem.

Uživatelské rozhraní a implicitní stylování aplikace bude respektovat designový jazyk „Material design“. Pro docílení tohoto vzhledu aplikace je nutné použít knihovny pro zpětnou kompatibilitu se staršími verzemi operačního systému Android. Níže uvedené zjednodušené návrhy jednotlivých obrazovek aplikace znázorňují rozmístění komponent a způsob navigace v aplikaci.



Obrázek 16: Zjednodušený návrh uživatelského rozhraní (zleva menu, slevové akce, kategorie produktů)

5.6 Konfigurace a rozšiřitelnost

Framework bude umožňovat nastavení URL katalogu, atributy animací, typ kešování a maximální počet stahovaných objektů (dodatečné načítání objektů při „scrollování“). Hodnoty těchto atributů nastaví vývojář v metodě *OnCreateBundle* aktivity, která dědí z třídy *CatalogueMainActivity*. Tato třída se stará o inicializaci všech potřebných objektů pro chod katalogu.

Změny implicitního chování či vzhledu bude vývojář schopen provést vytvořením tříd, které dědí z veřejných tříd frameworku (*AbstractCatalogueConnector, ProductDetail, Filter* atd.). Poté ve vývojářem definované třídě přepíše funkcionalitu vybraných metod a následně ve třídě *CatalogueMainActivity* nastaví framework tak, aby používal jeho implementaci tříd. Tímto způsobem může například ve třídě *ProductDetail* vytvořit novou komponentu, která bude automaticky frameworkem přidána na obrazovku s produktem. Tento model rozšiřitelnosti je zatím jen návrh a může se ještě značně změnit v závislosti na způsobu implementace Frameworku.

6 Implementace nabídkového systému

Tato kapitola se zaměřuje na technickou část práce. Popisuje implementační detaily vytvořené Android knihovny, její použití a možnosti customizace. Dalším tématem bude implementace ukázkové aplikace, na níž je demonstrováno využití knihovny a propojení s e-commerce systémem. Popíšeme si také použité knihovny třetích stran, které vzorová aplikace či samotná knihovna využívá.

6.1 Implementace knihovny StoreBuilder

Hlavním účelem knihovny StoreBuilder je zjednodušit práci vývojářům e-commerce aplikace na platformě Android. Ještě před její implementací byla zpracována analýza jak existujících univerzálních řešení, tak i oblíbených e-commerce aplikací konkrétních obchodů. Z analýzy vzešly základní požadavky, které e-commerce aplikace splňují. Jak již bylo zmíněno v návrhu, knihovna StoreBuilder se věnuje hlavně prezentaci produktů. Samotný nákupní proces vedoucí k dokončení objednávky není obsahem této knihovny, a to zejména kvůli odlišným přístupům k této problematice.

Knihovna StoreBuilder je hlavně sada komponent uživatelského rozhraní a metod pro snadnou komunikaci s konektorem e-commerce serveru. Zaměřuje se zejména na procházení katalogem, vyhledávání a prezentaci produktů. Vznik této knihovny doprovázel také vznik ukázkové aplikace, na níž je použito aplikační rozhraní a komponenty samotné knihovny. Aplikace demonstruje pouze jedno z možných využití knihovny. Koncová aplikace pak může vypadat jinak, nebo použije jen určitou část funkcionality knihovny. Vývojář by neměl být nijak výrazně limitován díky modularitě jednotlivých komponent knihovny.

Oproti návrhu se implementace poměrně liší. Zachován zůstal formát komunikace se serverem a přístup k implementaci konektoru. Změněn byl však přístup ke konfiguraci, customizaci a implementaci koncové aplikace. Oproti systému dědění aktivit a konfigurace systému je knihovna spíše sbírka komponent a funkcionality pro komunikaci se serverem. Tímto způsobem má vývojář větší volnost a více možností rozšíření výchozího chování. Customizace a rozšiřování poté probíhá formou dědění jednotlivých komponent bez nutnosti nahrazování celé funcionality. Knihovna také oproti návrhu nepokrývá funkcionalitu notifikací a konfiguraci produktových variant.

6.1.1 Datové struktury katalogu produktů

Z návrhu knihovny vyplývá, že katalog produktů je stromová struktura. Základní jednotkou je tedy uzel, který je reprezentován třídou *Node*. Tímto uzlem může být seznam produktových kategorií, seznam produktů či konkrétní produkt. Specifičtější typ uzlu je poté uzel seznamu (třída *Listing*), který reprezentuje seznam produktů či kategorií a je potomkem třídy *Node*. Ten poskytuje navíc kolekci potomků – seznam produktů či kategorií. Dále je také rozšířen o kolekci filtrů či typů řazení. Tyto třídy pak určují dle jakých atributů a podle jakých filtrů můžeme daný seznam řadit či filtrovat. Každá instance třídy *Node* nese svůj typ. Mezi výčet typů patří:

- **PRODUCT** – konkrétní produkt
- **CATEGORY_LISTING** – uzel kategorie nesoucí seznam podkategorií
- **PRODUCT_LISTING** – uzel kategorie nesoucí seznam produktů

Speciální typ uzlu je poté kořenový uzel, který identifikuje samotnou aplikaci. Ten lze vytvořit pomocí statické metody *Node.getRoot*. Kořenový uzel je identifikovatelný příznakem isRoot a používá se pro počáteční inicializaci aktuálního kontextu v katalogu.

Třída Node má pár předepsaných atributů, nicméně dovoluje vývojářům i tvorbu vlastních. Toho je dosaženo pomocí netypované hašovací tabulky, kde klíčem je textový řetězec nesoucí název atributu a hodnota poté netypovaná instance třídy *Object*. Knihovna se postará o automatické naplnění této kolekce a hodnoty lze získat pomocí metod *getObject* nebo *getString*. Mezi nejdůležitější rozhraní objektu patří tyto metody:

- **getID** – získání identifikátoru uzlu (povinný atribut);
- **getName** – získání jména uzlu (povinný atribut);
- **getType** – získání typu uzlu (povinný atribut);
- **isRoot** – identifikace, zda se jedná o kořenový uzel;
- **getImages** – získání kolekce URL odkazů na obrázky daného uzlu;
- **getObject** – získání netypovaného atributu z kolekce *customProperties;*
- **getString** – získání textové hodnoty atributu z kolekce *customProperties;*
- **getNodes** – získání uzlů potomků (pouze u třídy *Listing*);
- **getFilters** – získání kolekce filtrů (pouze u třídy *Listing*).

6.1.2 Práce s katalogem produktů

Jak již bylo popsáno v návrhu, základním úkolem knihovny je komunikovat s e-commerce serverem a prezentovat koncovému uživateli produkty. V této kapitole se zaměříme na prezentaci stromové struktury produktových kategorií a seznamu produktů.

Základním aplikačním rozhraním, se kterým budou vývojáři koncových aplikací pracovat, je třída *CatalogueProvider*. Přes tuto třídu budou vývojáři získávat data pro zobrazování seznamů produktů, kategorií či detailních pohledů na konkrétní produkt. Tato třída interně zajistí komunikaci se serverem pomocí formátu JSON, zpracování a převod na třídu Node (reprezentující uzel stromové struktury katalogu).

Na obrázku Obrázek 17: Získání obsahu katalogu je ilustrován základní přístup získání části produktového katalogu. Při vytvoření instance třídy *CatalogueProvider* je potřeba v konstruktoru specifikovat základní URL adresu serveru, který poskytuje aplikaci katalog. Metoda *getNode* provede asynchronní dotaz a výsledek je vrácen pomocí rozhraní *CatalogueHandler*. Parametr *node* je instancí třídy *Node*, pro niž chceme získat více informací (parametr typu *Node* typicky obsahuje pouze identifikátor a byl získán jako položka seznamu jiné části katalogu). V případě úspěšného zpracování má vývojář dostupnou plnou instanci dotazovaného uzlu v metodě *onDone*. V případě nezdaru je vrácen HTTP chybový kód v metodě *onError*. V případě chybného zpracování JSON formátu (nevalidní JSON či nevalidní formát odpovědi) je vrácena záporná hodnota.

```
new CatalogueProvider(BASE_URL).getNode(node, new CatalogueProvider.CatalogueHandler() {
    @Override
    public void onDone(Node n) {

    }

    @Override
    public void onError(int errCode) {

    }
});
```

Obrázek 17: Získání obsahu katalogu

Třída *CatalogueProvider* interně používá třídu *CatalogueParser* pro zpracování JSON formát a také třídu *DownloadTask*, která provádí asynchronní komunikaci se serverem. CatalogueParser převádí vstupní textový řetězec obsahující JSON data na instanci třídy *Node*, případně potomka této třídy *Listing* (uzel nesoucí seznam dalších uzlů). JSON je zpracováván pomocí Android tříd JSONArray a JSONOject. Zpracování probíhá ve dvou fázích – při té první se dle atributu *Type* rozhodne, o který typ uzlu se jedná, a v druhé fázi se již počítá s určitým formátem dat. Určité atributy (*ID*, *Name*, *Type* atd.) jsou povinné, nicméně vývojář je schopen jednotlivé uzly rozšířit o jakékoliv doplňující atributy, které jsou taktéž zpracovány a uloženy do kolekce *customProperties*. Vývojář pak může získat jejich hodnoty přes aplikační rozhraní třídy *Node*. V případě, že se vývojář rozhodne použít jiný formát než předepsaný formát JSON, může si parser nahradit vlastní implementací. Tento a jiné případy customizace si probereme ve zvláštní kapitole.

Přímou komunikaci se serverem zajišťuje třída *DownloadTask* rozšiřující knihovní Android Android *AsyncTask*. Ta asynchronně na pozadí stáhne data ze serveru a podobným přístupem jako *CatalogueProvider* je poskytne přes rozhraní s akcemi *onDone* a *onError* k následnému zpracování parserem. Pro samotnou HTTP komunikaci se serverem je využita knihovna **OkHttp**, kterou si popíšeme v kapitole o použitých knihovnách.

6.1.3 Zobrazování katalogu

V předchozí kapitole jsme si prošli získávání obsahu katalogu. Nyní se zaměříme na jeho zobrazení pomocí knihovny StoreBuilder. Pro zobrazování produktů použije vývojář Android komponentu *ListView* či *GridView*, kterou nastaví dle svých požadavků. Pro naplnění těchto komponent obsahem může využít třídu *ProductRowAdapter*, která dědí z Android třídy *BaseAdapter*. Vývojář vytvoří její instanci s parametrem *Listing*, který reprezentuje seznam produktů či kategorií. Poté tuto instanci nastaví jako adaptér do Android komponenty *GridView*. Knihovna se již postará o správné zobrazení položky seznamu a asynchronní načtení obrázku položky. Knihovna nabízí základní vzhled položky či produktu, nicméně vývojář je schopen toto chování upravit a zobrazit tak vlastní vzhled položky. Změnu chování těchto komponent si popíšeme v kapitole o customizaci.

Zobrazování detailu produktu lze implementovat mnoha způsoby a knihovna nepředepisuje žádná striktní pravidla. Pro náhledy obrázků lze využít komponentu *ImageSlider*. Pro zobrazení detailu obrázků s možností přibližování pak knihovna nabízí aktivitu *ImageDetailActivity*. Výběr počtu kusů pro přidání do košíku může být realizován pomocí komponenty *UnitCount*. Pro zobrazení popisu produktu (ve formátu HTML) je nejvhodnější použít Android komponentu *WebView* pro plnou podporu HTML a Javascriptové funkcionality. Výběr varianty produktu je pak plně na volbě vývojáře – použít lze například Android komponentu *Spinner*.

6.1.4 Komponenty uživatelského rozhraní

Mimo třídy pro získávání a zobrazování katalogu produktů knihovna poskytuje vývojářům komponenty, které jsou častou součástí aplikací pro elektronické obchodování. V této kapitole si krátce popíšeme funkci a účel těchto komponent. Jejich konkrétní použití si demonstrujeme v kapitole o implementaci vzorové aplikace.

UnitCount

Tato komponenta je určena pro zadávání a zobrazování počtu kusů produktu. Typické umístění této komponenty je před tlačítkem pro vložení produktu do košíku nebo pro úpravu a zobrazení počtu vybraných kusů v nákupním košíku. Komponenta zobrazuje tlačítka pro zvýšení a snížení počtu kusů a textové pole, do kterého může uživatel zadat kladné číslo pomocí klávesnice. Vývojář může tyto změny sledovat pomocí akcí.

SearchBox

Komponenta *SearchBox* je mírné rozšíření knihovny **Persistentsearch**, která poskytuje vývojáři možnost implementace vyhledávacího textového pole s našeptávačem výsledků.

OutlineTextView

Tato textová komponenta je rozšířením Android *TextView*. Oproti němu však zobrazuje okolo textu černý stín, který pomáhá s čitelností textu na předem neznámém pozadí. *OutlineTextView* se používá zejména nad obrázkem, jehož vzhled a barvu nejsme schopni určit, ale chceme zajistit dobrou čitelnost zobrazované informace bez nevzhledných pozadí rušících dojem obrázku.

FiltersView

FiltersView se používá pro zobrazení filtrů, jimiž chceme specifikovat určité produkty. Komponenta nabízí metody pro zobrazování dvojic jména filtru a výběrové komponenty. Tyto dvojice (filtry) jsou získány z definice filtrů třídy *Listing*. Zobrazovány jsou pod sebou a vývojář poté může získat všechny hodnoty výběru uživatele.

ImageSlider

Komponenta *ImageSlider* dědí z komponenty rozvržení RelativeLayout a stará se o vytvoření galerie obrázků. Galerií může uživatel listovat a ve spodní části zobrazuje indikátor aktuální strany. Tato galerie je určena pouze pro náhledy (nepodporuje přibližování) a poskytuje vývojářům akci kliknutí na zvolený obrázek.

ImageDetailActivity

Tato aktivita zobrazuje galerii obrázků v celoobrazovkovém zobrazení. Obsahuje indikátor aktuální stránky a podporuje i přibližování jednotlivých obrázků. Jako argumenty je jí potřeba dodat aktuální uzel produktu (*Node*) a index obrázku, na který má nastavit aktuální stránku galerie.

6.1.5 Fitrování a řazení

Produktové uzlu typu *Listing* podporují filtrování a řazení položek. Metodou *getFilters* je možné získat kolekci filtrů, kterou lze poté nastavit komponentě *FiltersView*. Samotná položka filtru pak obsahuje informaci o názvu, typu zobrazení (*Checkboxes*, *Dropdown*) a hodnot, kterých může filtr

nabývat. Komponenta *FiltersView* se pak postará o vytvoření příslušných prvků uživatelského rozhraní. Řazení je frameworkem vnímáno jako druh filtru a není nijak rozlišeno.

7 Implementace ukázkové aplikace

V rámci praktické části této práce vznikla kromě knihovny také ukázková aplikace. Ta se skládá z klientské a serverové části. Klientská část je určena ke vzorovému použití knihovny StoreBuilder – demonstruje využití jednotlivých komponent, metod a slouží i jako vzor pro customizaci jejích částí. Serverová část implementuje konektor mezi systémem pro zprávu obsahu Kentico a klientskou aplikací. V následujících kapitolách si popíšeme implementační detaily těchto částí aplikace.

7.1.1 Klientská část aplikace

Ukázková aplikace vychází ze základní šablony pro projekt Android studia. Aplikace pokrývá základní použití knihovny StoreBuilder a implementuje tedy prohlížení katalogu produktů. Jako zdroj dat posloužil konektor k systému pro zprávu obsahu Kentico. Pro testovací data byl použit Kentico ukázkový web zabývající se prodejem kávy.

Implementace pokrývá zobrazení seznamu kategorií, seznamu produktů, detailní pohled na produkt a prohlížení jeho obrázků. Dále také filtrování, řazení a vyhledávání produktů. Mimo funkcionalitu poskytovanou knihovnou je implementováno také zobrazení jednoduchého nákupního košíku. Proces nákupu počínající nákupním košíkem není touto aplikací pokryt a nebyl ani náplní této práce, a to zejména kvůli různosti přístupu k této problematice. Aplikace je určena pro demonstraci použití knihovny a nejsou zde implementovány např. chybové hlášení a indikace načítání obsahu.

7.1.2 Struktura aplikace

Vzorová aplikace je postavena pomocí jedné aktivity a tří fragmentů pro zobrazení jednotlivých případů použití – zobrazení seznamu produktu či kategorií, zobrazení detailu produktu a zobrazení nákupního košíku. Hlavní aktivita se stará o zobrazování fragmentů a celkovou navigaci v aplikaci.

Panel akcí (actionbar) je obsažen přímo v aktivitě, a je tedy sdílen mezi fragmenty. Toto řešení bylo vhodné zejména kvůli podobnému účelu panelu akcí napříč fragmenty. Na druhou stranu však přináší nutnost jeho korektní aktualizace při každém přechodu mezi fragmenty. Typickým příkladem je skrývání jednotlivých akcí – například při zobrazení košíku nechceme zobrazovat akci s ikonou lupy pro hledání v katalogu produktů. Při každém přechodu mezi fragmenty je zjištěna aktuální instance fragmentu a obsah panelu akci je aktualizován dle hodnot této instance. Při změně je aktualizována ikona navigace (tlačítko pro otevření postranního menu versus šipka indikující navigaci zpět), nadpis a zobrazení či skrytí ostatních akcí.

Jednotlivé fragmenty jsou zobrazovány pomocí Android komponenty *FrameLayout*. Přechody mezi nimi jsou realizovaný skrytím aktuálního fragmentu, jeho přidáním do zásobníku zpětné navigaci a poté přidáním nové instance fragmentu. Tímto způsobem je zajištěno setrvání aktuálního stavu fragmentu při zpětné navigaci. Uživateli bude tedy obnovena např. pozice v seznamu produktů po zpětném přechodu z detailu produktu.

7.1.3 Seznamu produktů a kategorií

O zobrazení seznamu produktů a kategorií se stará fragment *StoreFragment*. Instanci tohoto fragmentu získáváme pomocí statické metody *getInstance*, přijímající argument typu *Node*. Tímto způsobem jsme schopni do instance fragmentu vpravit parametry, které se poté využijí při vykreslování uživatelského rozhraní fragmentu. První instance fragmentu dostane jako parametr kořenový uzel (uzel samotné aplikace). Poté jsou již instance vytvářeny s parametrem uživatelova vybraného uzlu produktu či kategorie.

Při zobrazení *StoreFragment* fragmentu je vykonán asynchronní dotaz na server za účelem získání seznamu položek. Tento dotaz se uskutečňuje pomocí knihovní třídy *CatalogueProvider*. Ještě před uskutečněním samotného požadavku, který je plně v režii knihovny, je potřeba nakonfigurovat, na kterou adresu se má klient dotazovat. V následujícím úryvku kódu je demonstrováno ukázkové nastavení URL adres pro různé typy uzlů. Pro zjednodušení jsou adresy reprezentovány statickými hodnotami.

```
CatalogueProvider cataloguеProvider = new CatalogueProvider();
cataloguеProvider.setUrlProvider(new CatalogueProvider.URLProvider() {
    @Override
    public String getNodeUrl(Node node) {
        if (node.isRoot()) {
            return SampleGenerator.CATEGORIES_URL;
        } else {
            return SampleGenerator.PRODUCTS_URL;
        }
    }
});
```

Obrázek 18: Nastavení různých URL adres dle typu uzlu

Po vytvoření a nastavení instance *CatalogueProvider* již můžeme provést asynchronní získání položek ze serveru ilustrované na obrázku Obrázek 17. V případě úspěšného získání dat si v akci onDone přetypujeme uzel na typ *Listing* a ten použijeme pro vytvoření nové instance knihovní třídy *ProductRowAdapter*. Instanci poté nastavíme jako adaptér Android komponentě GridView, jež nám vykreslí daný seznam položek se základním vzhledem definovaným knihovnou. V případě, že má daná instance třídy *Listing* dostupné filtry, zobrazíme akci s vyvoláním dialogu filtru.

7.1.4 Filtrování produktů

Pro zobrazování filtru je určena komponenta FilterView definovaná v knihovně. Ukázková aplikace zobrazuje tuto komponentu v dialogu. Pro nastavení aktuální sady filtrů je využita metoda *addFilters* s parametrem filtrů, získaných z aktuálního uzlu tylu *Listing*. Po potvrzení dialogu pro nastavení filtrů jsou aktuální hodnoty filtru získány metodou *getValues*. Tyto hodnoty jsou poslány serveru, který vrátí upravený seznam produktů. Jednoduchý příklad filtru (dle typu zpracování kávy) je dostupný v sekci „Coffee“.

7.1.5 Vyhledávání produktů

Vyhledávání produktů je v ukázkové aplikaci implementováno pomocí komponenty *SearchBox*. Po kliknutí na ikonku lupy se v panelu akcí zobrazí box pro vyhledávání. Zajistí to metoda komponenty *revealFromMenuItem*. Při zadávání textu do pole je opakovaně spouštěna akce

onSearchTermChanged, ve které se za pomocí třídy *CatalogueProvider* provede asynchronní dotaz na server a výsledky jsou vloženy do našeptávače komponenty. Při kliknutí na položku je v akci *onResultClick* uživatel přesměrován na detail produktu.

7.1.6 Zobrazení detailu produktu

Produktový detail je realizován promocí fragmentu *DetailFragment*. Tento fragment je stejně jako *StoreFragment* vytvářen pomocí statické metody s parametrem vybraného uzlu. Stejným způsobem je i použit *CatalogueProvider* k získání detailních informací o produktu. Z panelu akcí jsou skryty nerelevantní akce vyhledávání a filtrování a nadpis je nastaven dle jména produktu. Po načtení informací ze serveru je inicializována galerie náhledů, která při kliknutí na obrázek spouští instanci *ImageDetailActivity* na příslušné stránce. Dále je zde zobrazen název a cena produktu. Pro akci přidání produktu do košíku je zde použita komponenta *UnitCount*. Po kliknutí na klasický Android *Button* je hodnota počtu kusů přečtena z komponenty a zavolána metoda pro přidání produktu do košíku. O zobrazení popisu produktu se stará Android komponenta *WebView*, která podporuje HTML formátování. Efektní pohyb obrázku vůči zbytku uživatelského rozhraní při „scrollování“ je realizován pomocí knihovny **ParallaxScrollView**.

7.1.7 Nákupní košík

Nákupní košík je v ukázkové aplikaci využívá z knihovny pouze komponentu *UnitCount* a statické metody z třídy *StoreUIHelper*, které slouží pro vytvoření položky v menu s indikací počtu kusů v nákupním košíku. Samotný nákupní košík je pouze pro ukázku realizován pomocí třídy *CartHolder*. Ta je implementována dle návrhového vzoru „singleton“ a poskytuje metody pro přidávání, odstraňování a modifikaci počtu kusů v košíku. Pro zobrazení je poté použita komponenta Android *ListView* a počty kusů lze měnit pomocí již zmíněné komponenty *UnitCount*.

7.2 Serverová část aplikace

Pro serverovou část ukázkové aplikace byla na Windows server nainstalována instance systému Kentico. Toto komplexní webové řešení se zabývá systémem pro zprávu obsahu, nástroji pro internetové obchodování a internetový marketing. Katalog produktů pro aplikaci byl použit z ukázkového webu Dancing Goat. Ukázkový web obsahuje pouze jednu úroveň kategorií, pod nimiž jsou již konkrétní produkty. Podpora systému Kentico pro internetové obchodování je daleko rozsáhlejší než to, co je jádrem knihovny *StoreBuilder*. Systém podporuje více měn, daňový výpočet, podmíněné slevy atd. Díky možnosti customizace a rozšíření knihovny by však bylo možné implementovat do koncové aplikace i další funkce.

Konektor mezi systémem Kentico a ukázkovou aplikací byl vytvořen za pomocí .NET frameworku. Projekt konektoru byl vytvořen v nástroji Visual Studio. Pro použití Kentico aplikačního rozhraní byly do projektu referencovány Kentico knihovny pomocí balíčkovacího nástroje NuGet. Komunikace s klientskou aplikací byla zprostředkována pomocí frameworku Web API. Tento framework je určen pro vývoj HTTP služeb s REST (Representational State Transfer) architekturou rozhraní. Framework podporuje více typů formátu dat, nicméně pro naše účely použijeme formát JSON.

7.2.1 Zprostředkování katalogu produktů

Třída *ProductController* zprostředkovává pomocí jeho metod katalog produktů. Jména metod jsou automaticky mapovány na příslušnou URL adresu. Parametry metod odpovídají parametrům URL adresy. Produkty jako takové jsou v systému uloženy jakou soubor informací v různých databázových tabulkách. Uzel stromové struktury stránky je uložen v tabulce *Node*, obsah a základní informace poté v tabulce *Document* (jeden uzel může být tvořen např. více jazykovými verzemi). Informace o produktu je obsažena v tabulce *SKU*. Každý produktový typ má navíc svoji vlastní tabulku, v níž jsou uloženy vlastnosti specifické pro daný typ (např. typ zpracování kávy). Veškeré tyto informace jsou spojeny pomocí Kentico API. Následující úryvek kódu demonstruje získání produktů dle identifikátoru kategorie.

```
var query = DocumentHelper.GetDocuments()
                .Path("/Store/", PathTypeEnum.Children)
                .WhereEquals("NodeParentID", categoryID)
                .OnSite("DancingGoat");
```

Obrázek 19: Získání produktů pomocí Kentico API

Návratový typ reprezentuje pouze dotaz do databáze. Po zavolání metody ToList se tento dotaz provede a máme k dispozici kolekci konkrétních uzlů (v našem případě produktových uzlů). Díky metodám uzlů s prefixem „Where“ můžeme dotaz dále specifikovat. Tento přístup uplatňuje například implementace produktového filtru. Na straně serveru byly s použitím Kentico API implementovány tyto metody:

- **Index** (adresa *api/product*) – Metoda získává seznam produktů pod danou kategorií. Pokud je zadán i parametr *filter*, provede filtrování výsledků dle zadané hodnoty. Návratovým typem uzlu je seznam produktů.
- **Search** (adresa *api/product/search*) – Metoda vyhledává mezi všemi produkty dle zadaného řetězce. Navrácen je seznam produktů obsahujících v názvu zadaný řetězec.
- **Detail** (adresa api/product/detail) – Metoda vrací detailní pohled na produkt dle zadaného identifikátoru. Návratovým typem je konkrétní produkt obsahující detailní popis ve formátu HTML a seznam adres obrázků.
- **Categories** (adresa *api/product/categories*) – Tato metoda vrací seznam produktových kategorií.

7.3 Použité knihovny

Tato kapitola obsahuje výčet použitých knihoven třetích stran. Popsáno bude jejich použití a účel v knihovně StoreBuilder. Zvoleny byly pouze volně šiřitelné knihovny a obsah jejich licenčních ujednání je dostupný na přiloženém CD.

Picasso

Knihovna Picasso je určena pro pohodlné načítání obrázků z různých zdrojů. V této práci je knihovna použita zejména pro načítání obrázků z internetových zdrojů a jejich automatické kešování. Vývojář koncové aplikace si poté může načítání touto knihovnou upravit.

OkHttp

Tato knihovna zjednodušuje komunikaci klientské aplikace s webovým serverem pomocí protokolu HTTP. Dostupné jsou zde metody pro jednoduchou stavbu dotazu a zpracování odpovědi. Knihovna StoreBuilder toto knihovnu využívá pro asynchronní komunikaci se serverem za použití GET HTTP požadavků.

PersistentSearch

Tato komponenta je určena pro zobrazení vyhledávacího boxu v aplikaci. Podporuje našeptávač výsledků a je možné ji použít na panelu akcí. Knihovna StoreBuilder tuto komponentu lehce upravuje pro dané použití a v ukázkové aplikaci je komponenta využita pro vyhledávání produktů.

PhotoView

Za pomocí této komponenty implementuje knihovna StoreBuilder galerii obrázků. Komponenta umožňuje uživateli přibližovat obrázek pomocí klasických dotykových gest.

ViewPagerIndicator

Tato komponenta byla spolu s *PhotoView* použita pro implementaci galerie obrázků. Komponenta se stará o zobrazení indikátoru aktuální stránky v Android komponentě *ViewPager*. Podporovány jsou různé vzhledy a tvary indikací.

ParallaxScrollView

Tato komponenta byla použita pouze v ukázkové aplikaci pro efektní „scrollování“ obrázku vůči zbytku obrazovky detailu produktu. Jedná se o komponentu rozvržení, která v základním nastavení zajistí posuvný efekt první vnořené komponenty oproti zbytku komponent.

7.4 Customizace knihovny

Knihovna StoreBuilder nabízí vývojářům zejména komponenty, s jejichž pomocí lze snadným způsobem implementovat zobrazování katalogu produktů. Ne všem však jejich výchozí chování a vzhled bude vyhovovat. Častým požadavkem bude také rozšíření základních funkcí. V této kapitole si probereme základní přístupy k úpravám chování knihovny pro typické požadavky.

Rozšíření protokolu

Typickým požadavkem bude určitě rozšíření protokolu o specifické atributy – například cena před slevou pro produkt. Pokud se jedná o primitivní typ, vývojář může z instance *Node* získat netypovanou hodnotu daného atributu. Pokud se však jedná o složitější datový typ, knihovna ho uloží jako nezpracovaný řetězec formátu JSON. Ten je poté možno zpracovat dodatečně z instance třídy *Node*. Pokud bychom však chtěli pracovat s typovanými atributy, je zapotřebí customizovat samotný parser. Tento přístup si probereme v příští kapitole.

Rozdílný formát dat

Může nastat situace, kdy se vývojář rozhodne psát konektor na straně klienta. Typicky je tak učiněno, pokud e-commerce systém obsahuje rozhraní pro získávání produktů nebo konektor na straně serveru poskytuje data v jiném než předepsaném JSON formátu. Typickým postupem při této customizaci je vytvořit novou třídu dědící z třídy *CatalogueParser*. Vývojář poté přepíše metodu *getNode*, která

zpracuje vstupní formát na instanci třídy *Node*, nebo libovolného potomka této třídy. Posledním krokem bude nastavení nového parseru do třídy *CatalogueProvider* pomocí metody *setParserProvider*. Tento krok ilustruje níže uvedený obrázek.

```
cataloguеProvider.setParserProvider(new CatalogueProvider.ParserProvider() {
    @Override
    public CatalogueParser createParser(String data) {
        return new MyXMLParser(data);
    }
});
```

Obrázek 20: Customizace parseru

Pokud jsme si již vytvořili svůj parser, můžeme rozšířit i samotnou třídu uzlu a v metodě parseru *getNode* vracet instanci nové třídy dědící ze třídy *Node*. Tímto způsobem jsme schopni vytvořit vlastní typované atributy pro náš specifický typ uzlu.

V případě, že si nevystačíme s jednoduchými GET požadavky, musíme implementovat samotný *CatalogueProvider*. V tomto případě již nemá cenu vytvářet zděděné třídy a vývojář si bude muset konektor vracející instance tříd *Node* napsat sám – například s využitím **OkHttp** knihovny.

Změna vzhledu položky katalogu

Další z typických požadavků na customizaci je změna vzhledu pro položku katalogu. Vývojář si může napsat úplně vlastní adaptér, ale pokud chce provést pouze malou změnu chování či vzhledu, tak může využít dědičnosti a přepsat metodu *getViewForNode* pro třídu *ProductRowAdapter*. Na základě hodnoty či typu daného uzlu poté může změnit vzhled či způsob načítání obrázku. Práce s knihovnou **Picasso** je zapouzdřena v metodě *loadImageInto*, kterou může vývojář v případě potřeby využít. Pokud chce vývojář změnit způsob načítání obrázku nebo využít některou z možností, kterou nabízí knihovna Picasso, je možné v adaptéru přepsat metodu *picassoModifier*. Ta ve výchozím *ProductRowAdapter* adaptéru vrací nezměněný objekt. Vývojář poté může objektu Picasso nastavit vlastní hodnoty a upravit tak původní chování. Implementace modifikovaného adaptéru pro zobrazování produktů je demonstrována na níže uvedeném obrázku.

```
public class CustomRowAdapter extends ProductRowAdapter {

    public CustomRowAdapter(Context c, Listing listing) {
        super(c, listing);
    }

    @Override
    public View getViewForNode(Node node) {
        if(node.getString("Type").equals("MyType"))
            return getMyCustomView();

        return super.getViewForNode(node);
    }

    @Override
    public RequestCreator picassoModifier(RequestCreator p) {
        return p.placeholder(android.R.drawable.ic_dialog_alert);
    }
}
```

Obrázek 21: Customizace seznamu produktů

7.5 Testování aplikace a knihovny

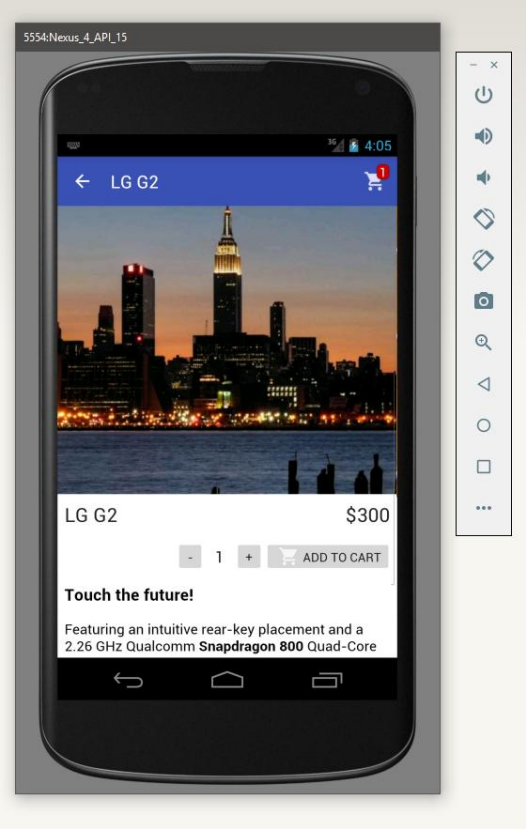
Vývoj knihovny doprovázel také vývoj ukázkové aplikace. Ta sloužila mimo demonstrační účely také jako testovací nástroj knihovny. Během vývoje se na ukázkové aplikaci testovalo nejen její použití, ale také možnosti její customizace.

Knihovna vznikala dříve než serverová část ukázkové aplikace. O vytváření testovacích dat se původně starala třída SampleGenerator, která vytvářela různé instance typů *Listing* a *Node*. Později při vývoji funkcí pro komunikaci se serverem byl tento generátor nahrazen statickými JSON daty. Pro jejich uložení byla použita služba GitHub gists. Ukázkové JSON soubory byly tedy veřejně dostupné pod URL adresou. Díky možnosti editace těchto JSON souborů v prohlížeči bylo tedy možné testovat získávání dat ze serveru a jejich následné zpracování.

Pro testování uživatelského rozhraní a funkčnost vzorové aplikace byl použit emulátor dodávaný v Android SDK. Vytvořeno bylo více instancí jednotlivých emulátorů, aby bylo pokryto co největší množství Android verzí a různých velikostí i rozlišení displeje. Knihovna a samotná aplikace podporuje Android verze 4.0.1 a vyšší. Odolnost aplikace vůči chybám připojení k internetu bylo pak testováno pomocí fyzického Android telefonu Samsung Galaxy Note 5.

Funkčnost vybraných komponent knihovny byla také pokryta automatickými jednotkovými testy. Využit byl testovací framework **JUnit**, který je do prostředí Android studia integrován, a testy byly spouštěny přímo na cílové platformě v emulátoru. Tyto testy byly hlavně použity pro testování zpracování formátu JSON a rozhraní datových struktur katalogu.

Serverová část konektoru se systémem Kentico byla vyvíjena zcela odděleně. Pro základní testování konektoru byl použit internetový prohlížeč. Pro pokročilejší komunikaci pak sloužil doplněk pro prohlížeče **Postman**, kterým byly otestovány složitější dotazy na server.



Obrázek 22: Testování aplikace pomocí emulátoru a testovacích dat

8 Závěr

V rámci této práce byl navržen a implementován framework sloužící pro zjednodušení vývoje nativních aplikací na platformě Android, jež nabízejí produkty zákazníkům. Hlavním účelem frameworku je zejména zobrazování produktů s možností vyhledávání, filtrování či řazení. Framework byl navržen s ohledem na jeho rozšíření, změnu implicitního chování nebo vzhledu. Vývojář koncové aplikace si bude mít možnost vybrat, zda implementuje aplikační rozhraní na straně serveru za použití implicitního konektoru na straně klienta, nebo konektor implementuje na straně klienta za použití stávajícího rozhraní serveru.

Framework je zatím navržen a implementován jen pro zobrazování katalogu, avšak do budoucna ho bude určitě vhodné rozšířit i o nástroje umožňující implementaci nákupního procesu. Ten se může značně lišit v závislosti na použitém e-commerce systému, lokalitě podnikání či typu nabízeného zboží. I přesto bude možné nabídnout vývojářům nástroj, který se postará např. o zobrazování nákupního košíku, vyplňování adres či výběr způsobu doručení zboží.

Dalšími možnostmi jeho vylepšení je implementace nových komponent (např. hodnocení produktu, komentáře) či navržení modelu pro konfiguraci produktových variant. Vhodným krokem by bylo také zveřejnit knihovnu jako projekt s otevřeným zdrojovým kódem, a získat tak zpětnou vazbu od vývojářů, kteří by toto knihovnu použili. Vítaným vylepšením by byl určitě také jednodušší způsob stylování komponent frameworku. Ideálně by framework mohl podporovat tvorbu šablon vzhledu.

Literatura

[1] LIBOR ŠVADLENKA, Radovan Madleňák. Elektronické obchodování. Vyd. 1. Pardubice: Institut Jana Pernera, 2007. ISBN 80-865-3040-X.

[2] DVOŘÁK, Jiří. Elektronický obchod: studijní text pro kombinované studium. Vyd. 1. Brno: VUT, 2004, 78 s. ISBN 80-214-2600-4.

[3] REYNOLDS, Janice. *The complete e-commerce book: design, build*. 2nd ed. San Francisco: CMP Books, c2004, v, 374 s. ISBN 15-782-0312-0.

[4] Dashboards. *Android developers* [online]. [cit. 2015-01-20]. Dostupné z: http://developer.android.com/about/dashboards/index.html

[5] Android NDK. *Android developers* [online]. [cit. 2015-01-20]. Dostupné z: https://developer.android.com/tools/sdk/ndk/index.html

[6] Android Studio Overview. *Android developers* [online]. [cit. 2015-01-20]. Dostupné z: http://developer.android.com/tools/studio/index.html

[7] Android Architecture. *Tutorialspoint*. [online]. [cit. 2015-01-20]. Dostupné z: http://www.tutorialspoint.com/android/android_architecture.htm

[8] Android. *Android developers* [online]. [cit. 2015-01-20]. Dostupné z: http://www.android.com

[9] Introducing ART. *Android developers* [online]. [cit. 2015-01-20]. Dostupné z: https://source.android.com/devices/tech/dalvik/art.html

[10] Mobicart tour. *Mobicart* [online]. [cit. 2015-01-20]. Dostupné z: http://www.mobi-cart.com/tour.jsp

[11] How it works. *Mob Ecommerce* [online]. [cit. 2015-01-20]. Dostupné z: http://www.mobecommerce.net/mobile-ecommerce/iphone-android-architecture.php

[12] Custom Components. *Android developers* [online]. [cit. 2016-05-15]. Dostupné z: http://developer.android.com/guide/topics/ui/custom-components.html

[13] Activities. *Android developers* [online]. [cit. 2016-05-15]. Dostupné z: http://developer.android.com/guide/components/activities.html

[14] Fragments. *Android developers* [online]. [cit. 2016-05-15]. Dostupné z: http://developer.android.com/guide/components/fragments.html

[15] Storage Options. *Android developers* [online]. [cit. 2016-05-15]. Dostupné z: http://developer.android.com/guide/topics/data/data-storage.html

[16] Caching Bitmaps. *Android developers* [online]. [cit. 2016-05-15]. Dostupné z: http://developer.android.com/training/displaying-bitmaps/cache-bitmap.html

[17] Redundant Downloads are Redundant. *Android developers* [online]. [cit. 2016-05-15]. Dostupné z: http://developer.android.com/training/efficient-downloads/redundant_redundant.html

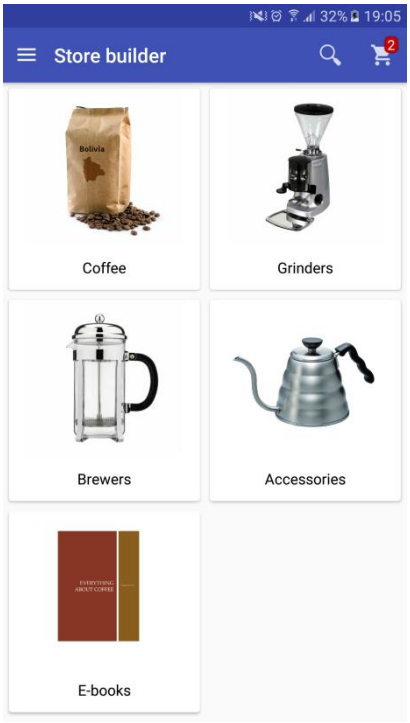
[18] *Material design* [online]. [cit. 2016-05-15]. Dostupné z: https://www.google.com/design/spec/material-design/

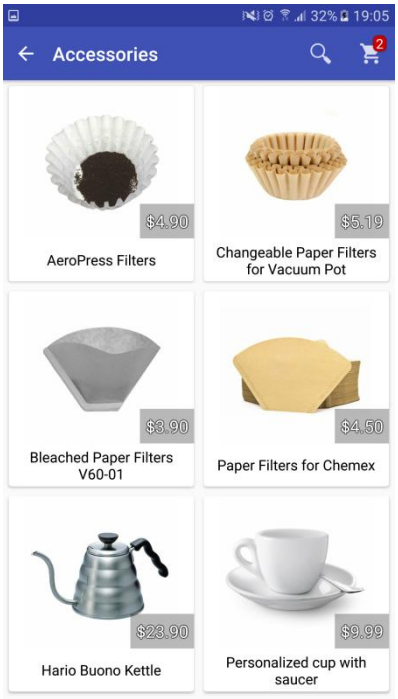
[19] Animations. *CodePath* [online]. [cit. 2016-05-15]. Dostupné z: https://guides.codepath.com/android/Animations

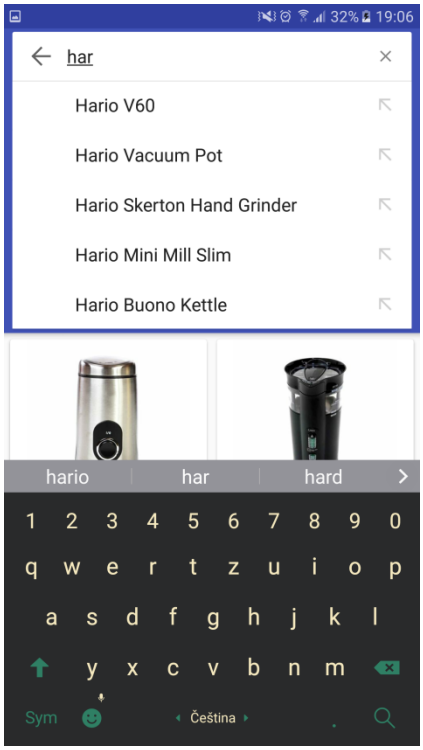
[20] Property Animation. *Android developers* [online]. [cit. 2016-05-15]. Dostupné z: http://developer.android.com/guide/topics/graphics/prop-animation.html

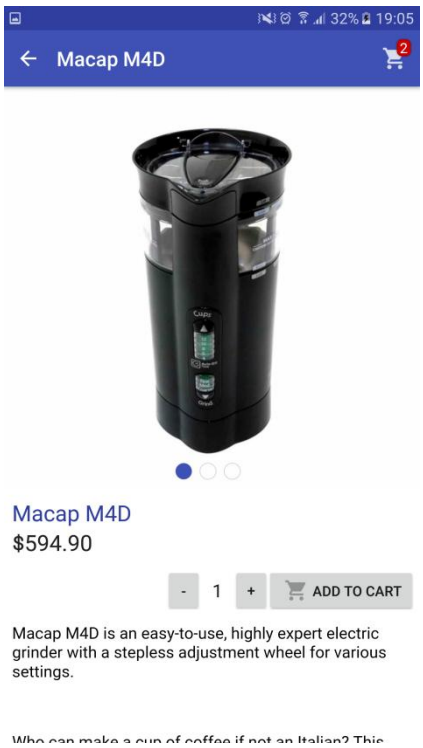
Seznam příloh

Příloha 1. Snímky obrazovky aplikace









Příloha 2. Obsah CD

Na přiloženém CD se nacházejí následující adresáře a soubory:

- /StoreBuilder – zdrojové soubory s projektem Android studia
- /Dist – instalační balíček aplikace
- /DP.pdf – diplomová práce ve formátu PDF
- /Connector – serverová část konektoru vzorové aplikace
- /licences.txt – seznam licencí použitých knihoven