VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

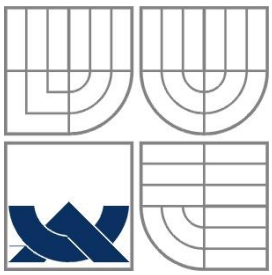
# NÁSTROJ PRO TRANSFORMACI E-SHOPŮ DO MOBILNÍCH ZAŘÍZENÍ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

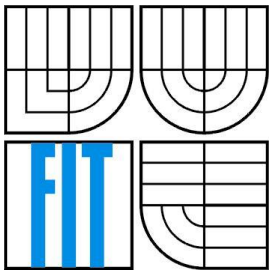
AUTOR PRÁCE
AUTHOR

ZDENĚK VÍDEŇSKÝ

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

# NÁSTROJ PRO TRANSFORMACI E-SHOPŮ DO MOBILNÍCH ZAŘÍZENÍ

A TOOL FOR TRANSFORMATION OF E-SHOPS INTO MOBILE DEVICES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ZDENĚK VÍDEŇSKÝ

VEDOUCÍ PRÁCE
SUPERVISOR

ING. VLADIMÍR BARTÍK, Ph.D.

BRNO 2015

# Abstrakt

Tato bakalářská práce se zabývá transformací dat z e-shopů do mobilní aplikace. Mým cílem je vytvořit komplexní systém pro vytvoření nativní mobilní aplikace existujícího e-shopu podle zadaných údajů od uživatele. Systém se skládá ze serverové a klientské aplikace. Na serveru běží webová aplikace napsaná v ASP .NET frameworku, která se stará o zpracování údajů zadaných uživatelem, o naplnění databáze produkty a kategoriemi na základě poskytnutého XML feed. Poskytuje také webové API pro komunikaci s klientem. Klient je nativní mobilní aplikace běžící na platformě Xamarin. Tato aplikace je psána multiplatformně pomocí Xamarin.Forms. Výsledkem této práce je ale zatím jen verze pro systém Android. Z této aplikace je možné procházet katalogem produktů a vytvářet objednávky, které se ukládají na server.



# Abstract

This bachelor thesis is focused on transformation of data from e-shops into the mobile application. My goal is to create a complex system for creating the native mobile app for existing e-shop based on the data from a user. The system consists of server-side and a client-side applications. On the server there is a running web app written in the ASP .NET framework, which cares of the processing data entered by a user and of filling database by products and categories on the base of provided XML feed. The server-side application also provides the web API for communication with a client. The client is a native mobile application running on Xamarin platform. This application is written multiplatformly with Xamarin.Forms, the result of this tesis is the only application for Android system. With this application the user is able to browse products and create the orders which are stored on server.



# Klíčová slova

Mobilní e-shop, .NET Framework, Mono, Xamarin, zpracování XML, webová aplikace, Xamain.Forms

# Keywords

Mobile e-shop, .NET Framework, Mono, Xamarin, zpracování XML, webová aplikace, Xamain.Forms

# Citace

Vídeňský Zdeněk: Nástroj pro transformaci e-shopů do mobilních zařízení, bakalářská práce, Brno, FIT VUT v Brně, 2015

# Nástroj pro transformaci e-shopů do mobilních zařízení

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Vladimíra Bartíka, Ph.D.
Další informace mi poskytli Ing. Petr Křenek a RNDr. Roman Stoklasa.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Zdeněk Vídeňský
18. května 2015

## Poděkování

Děkuji vedoucímu práce Ing. Vladimíru Bartíkovi, Ph.D. za konzultace, které mi pomohly při formování konečné podoby aplikace. Dále děkuji Ing. Petru Křenkovi a RNDr. Romanu Stoklasovi za poskytnutý vzorek dat a odborné rady, které napomohly ke zdárné implementaci aplikace.

# Obsah

# 1 Úvod

Obchodování po internetu prostřednictvím elektronických obchodů (e-shopů) je dnes jeden z nejrozšířenějších způsobů obchodování vůbec. S příchodem chytrých telefonů se v posledních letech více a více používají na procházení webových stránek a také k nakupování právě telefony. Dokonce podle jednoho z průzkumů společnosti Comscore 4 z 5 zákazníků používají k nakupování právě svůj mobilní telefon [1]. Téměř všechny e-shopy mají podporu zobrazování také pro mobilní zařízení, ale ve světě chytrých telefonů hrají prim právě nativní aplikace. Právě velká popularita nativních aplikací vedla k myšlence vytvořit nástroj, který dokáže již fungující e-shop na základě seznamu produktů v určitém formátu transformovat do nativní aplikace. Tento nástroj sestává jak ze serverové části, tak z klientské části, přičemž serverovou část tvoří webová aplikace, která se stará o aktualizaci databáze. Klientskou část tvoří nativní aplikace, která komunikuje se serverem a stahuje si aktuální položky z databáze. I když je celá nativní aplikace psána multiplatformně pro tři hlavní mobilní operační systémy (Android, iOS a Windows Phone), byla hlavně vytvořena a testována zatím jen pro systém Android, ze které také pochází všechny ukázky.

V kapitole 2 si popíšeme tvorbu webových aplikací společně s jejich strukturou a podíváme se také na MVC architekturu. Dále se poučíme o programovacích jazycích, ve kterých lze webové aplikace naprogramovat. Společně s tím si porovnáme webové a nativní aplikace na mobilních zařízeních a vysvětlíme si také, jak se na nich dají prezentovat data. V poslední řadě si řekneme něco o obecném konceptu e-shopů.

Kapitola 3 se už konkrétně věnuje zdroji dat pro náš systém. Popíšeme si jeden ze způsobů, jak získat databázi produktů a kategorií z jednotného formátu a jak by tento formát měl vypadat. Zanalyzujeme si požadavky na aplikaci v kapitole 4 a navrhneme si databázové tabulky. Poté si naimplementujeme serverovou část našeho systému, který bude naše databázové tabulky plnit daty. Celá kapitola 5 pojednává o platformě Xamarin na tvorbu nativních aplikací pro Android, iOS a Windows Phone. Podrobnému průběhu tvorby mobilní aplikace se věnuje kapitola 6. V kapitole 7 najdeme popis fungování systému, ověření funkčnosti a ukážeme si konečnou mobilní aplikaci s konkrétními daty. V kapitole 8 si celou práci shrneme a promluvíme si o dalším možném pokračování tohoto projektu.

# 2 Tvorba webových aplikací

Webové aplikace jsou aplikace poskytované uživatelům z webového serveru, kde je aplikace uložena, prostřednictvím počítačové sítě. Počítačová síť může být vnitropodniková (intranet) nebo, a to je ta nejčastější varianta, globální síť Internet.

Webová aplikace je uložena a běží na webovém serveru. Tento server je připojen k databázi, se kterou komunikuje. Uživatel komunikuje s webovým serverem pomocí webového prohlížeče protokolem HTTP.


*Obrázek 2.1: Fungování webové aplikace [2]*

## 2.1 Webový prohlížeč jako klient

Jedna z hlavních výhod v použití webových aplikací spočívá v tom, že uživatel si nemusí kvůli každé aplikaci instalovat zvláštního klienta, který umí s aplikací komunikovat. Jako klient pro webové aplikace se používá webový prohlížeč, který každý operační systém už buď obsahuje, nebo může být doinstalován jiný. Webový prohlížeč je tzv. tenký klient. Tenký klient je počítač nebo počítačový program, který je při plnění své funkce závislý na jiném počítači (serveru). Webový prohlížeč komunikuje pomocí HTTP protokolu s webovým serverem, přijatá data pomocí značek (HTML, XML, atd.) zformátuje a zobrazí uživateli na obrazovku počítače. Nejznámější webové prohlížeče jsou např. Google Chrome, Mozilla Firefox, Internet Explorer, Opera nebo Safari.

## 2.2 Struktura webových aplikací

Když se bavíme o webové aplikaci, určitě nemyslíme obyčejnou jednostránkovou aplikaci, která nám jen zobrazí údaje napsané ve zdrojovém textu. V dnešní době se od webové aplikace požaduje, aby měla také přístup k databázi, kam bude ukládat data, a aby měla vlastní aplikační logiku. Proto je nejlepší tyto jednotlivé komponenty rozdělit do vrstev. Nejvíce se využívá třívrstvá architektura.

## 2.3 MVC architektura

Jde o architekturu s trojúhelníkovou topologií a zároveň o jednu z nejoblíbenějších architektur pro webové aplikace. Tato architektura je i součástí některých webových frameworků, jakými jsou např. Zend, Nette nebo Ruby On Rail.

Architektura MVC dělí aplikaci na 3 logické části tak, aby je šlo upravovat samostatně a dopad změn byl na ostatní části co nejmenší. Tyto části jsou Model, View a Controller, jejichž počáteční písmena tvoří název této architektury [3].

### 2.3.1 Model

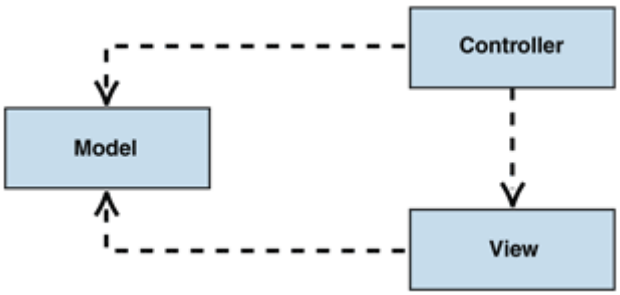
Model spravuje chování a data aplikační domény, odpovídá na dotazy ohledně jeho stavu (nejčastěji od View) a odpovídá na instrukce o změně jeho stavu (nejčastěji od Controlleru). Spravuje si svůj vlastní stav a výstupem je pevně dané rozhraní.

### 2.3.2 View

View zobrazuje výsledek požadavku (HTML, CSS). Neobsahuje žádnou aplikační logiku. Například když uživatel někam klikne myší, některou z metod, která je ale aplikovaná v Controlleru.

### 2.3.3 Controller

Controller má na starosti tok událostí v aplikaci a obecně i aplikační logiku. Metody Controlleru mohou být volány z View a tyto metody mohou měnit obsah modelu.




*Obrázek 2.2: MVC architektura [5]*

Všimněte si, že View a Controller přímo závisí na Modelu. Model nezávisí ani na Controlleru a ani na View. Tohle je klíčová výhoda takového rozdělení aplikace. Toto rozdělení potom dovoluje, aby byl Model sestaven a testován samostatně a nezávisle na vizuální prezentaci [4].

## 2.4 Programování webových aplikací

V dnešní době lze naprogramovat webovou aplikaci ve většině populárních programovacích jazyků. Existují například knihovny pro jazyk C++ nebo podpora pro jazyk Java. Existují také přímo frameworky pro vytváření webových aplikací, jako například MVC framework pro ASP .NET, kde se programuje v jazyce C#. Nejoblíbenějším jazykem pro tvorbu webových aplikací ale stále zůstává skriptovací programovací jazyk PHP, pro který je vytvořeno spousta frameworků, jako například český Nette Framework, Zend, Yii apod.

Po zvolení programovacího jazyka, případně frameworku, je také potřeba zvolit databázi, do které bude naše aplikace ukládat data. Takových databází, nebo přesněji Systémů řízení báze dat (SŘBD), což je softwarové vybavení, které zajišťuje práci s databází, je na výběr spousta typů, jako např. relační SŘBD Oracle, MySQL, MS SQL, SQLite atd.

## 2.5 Webové aplikace v mobilních zařízeních

Přestože, ve světě chytrých telefonů mají výsadní postavení nativní aplikace, nemalé procento uživatelů prochází přesto webové stránky pomocí mobilního zařízení.

### 2.5.1 Nativní a webové aplikace

Při vývoji aplikace pro mobilní zařízení se nevyhneme rozhodnutí zda zvolit nativní, či webovou aplikaci. Webové aplikace jsou přístupné prostřednictvím internetového připojení a webového prohlížeče v zařízení. Jedná se vpodstatě o weby, které nabízejí funkcionalitu aplikací.

Nativní aplikace jsou vyvinuty specificky pro danou platformu (Android, iOS, Windows Phone) a jsou do zařízení nainstalovány podobně jako aplikace na stolním počítači. Aplikace tohoto typu jsou distribuovány nejčastěji pomocí tržišť aplikací zaměřených na jednotlivé platformy. Například aplikace pro iPhone jsou k dispozici přes AppStore, na Android na Google Play a Windows Phone přes Windows Phone Store.

Hlavní nevýhoda nativních aplikací spočívá v tom, že jsou naprogramovány jen na určitou platformu. Například aplikace pro iOS jsou programovány v Objective-C, pro Android v Java a Windows Phone v C#. Právě nutnost programování aplikací na každou platformu zvlášť může být také finančně a časově náročné. Další nevýhodou je například manuální aktualizace. Když vyjde nová verze aplikace, je potřeba ji nejdřív aktualizovat, aby mohla nabídnout nové funkce. Výhoda webových aplikací je v tom, že není programována pro konkrétní platformu a její aktualizace se děje prakticky okamžitě. Ovšem nativní aplikace mohou nabízet příjemnější uživatelské rozhraní a jsou i celkově rychlejší.

Nevýhodou webových aplikací je fakt, že jsou interpretovány pomocí webových prohlížečů v mobilních zařízeních. Tato nutnost interpretace znamená, že je aplikace omezena schopnostmi příslušného stroje, jako příklad mohou být například nabízená oznámení, fotoaparát, mikrofon nebo kontakty uživatele.

Pokud máme již existující webovou aplikaci a chceme vytvořit její verzi pro mobilní zařízení, máme tedy na výběr, jestli uděláme nativní aplikaci, která bude využívat naše API, nebo upravíme tuto webovou aplikaci i pro správné zobrazení na mobilních zařízeních. V tomto případě máme několik možností, jak webovou aplikaci upravit.

První možností je neudělat nic. Pokud máme stránky, které nestahují moc dat a které mají dobře zobrazitelný design i na mobilních aplikacích, nemusíme dělat s webovou aplikací vůbec nic. Nové typy smartphonů již umožňují velmi snadnou navigaci i na těch nejsložitějších stránkách.

Druhou možností je transformace. V tomto případě přichází na řadu responsive design. Pojem „responsive design“ reprezentuje zvláštní umění využití dotazů na média jazyka CSS, dynamických mřížek a dynamických obrázků k přizpůsobení webu rozlišení obrazovky zařízení (nebo okna prohlížeče) uživatele a jeho navržení tak, aby nabízel co nejlepší možné rozložení pro jakékoliv rozlišení.

Třetí možností je vytvořit úplně oddělený web určený pro mobilní uživatele. Podobné weby jsou obvykle součástí subdomény `m.` nebo `mobile.` Potom je potřebný nějaký detekční mechanismus, který nás automaticky převede na tyto stránky pro mobilní zařízení.

Prezentovat data na mobilních zařízeních lze tedy buď přímo z webové aplikace za použití mobilního prohlížeče, nebo s využitím nativní aplikace, která komunikuje ze serverem na základě API. Já jsem se pro můj systém rozhodl jít cestou nativní aplikace, která stahuje data ze serveru [5].

## 2.6 E-shop

E-shop neboli elektronický obchod je speciální webová aplikace sloužící ke zprostředkování obchodních transakcí na internetu, většinou v oblasti B2C.[1]

Základem e-shopu je katalog produktů, které jsou obvykle řazeny v určitých kategoriích. V databázi jsou o každém produktu uložené různé informace. Tyto informace jsou různé a záleží na konkrétním e-shopu. Základní informace, které ale nesmí na žádném e-shopu chybět, jsou: název, cena a dostupnost na skladě. Nepostradatelnou součástí e-shopu bývá také vyhledávání, ať už na základě fulltextu[2] nebo na základě třídicích filtrů.

Jakmile si uživatel e-shopu vybere svůj produkt, dochází k nákupnímu procesu. Jeho prvním krokem je přidání produktu do tzv. košíku, speciální stránky, kde se shromažďují vybrané produkty. Poté je třeba produkty umístěné v košíku objednat, k čemuž jsou potřeba kontaktní údaje ze strany uživatele [6].

Po dokončení nákupního procesu je vystavená faktura na dané zboží a na základě platebních podmínek a způsobu převzetí je poté zboží předáno zákazníkovi.

### 2.6.1 První e-shopy

První e-shopy se začaly objevovat v USA v 90. letech 20. století. Internetové obchody „jak je známe dnes“ začínají vznikat v roce 1994 a to hned z několika důvodů. Marketingová oddělení si uvědomila, že internet je médium zpřístupněné milionům lidí po celém světě. Dalším podnětem bylo, že v tomto roce bylo vyvinuto SSL šifrování, které umožňuje zabezpečit relace přenášející data o kreditních kartách.

Za průkopníky v odvětví elektronické komerce lze bezesporu označit společnosti Amazon a eBay (obě vznikly v roce 1994) [7].

[1] Nejrozšířenější model internetového podnikání. Zkratka pochází z anglického Business to Customer. Jedná se tedy o přímý prodej koncovým zákazníkům.

[2] Obecné označení pro vyhledávače fungující na základě porovnání fráze se všemi ostatními slovy v daném dokumentu.

# 3 Zdroje dat pro e-shop

V této kapitole se podíváme na způsob importu produktů z katalogu existujícího e-shopu do databáze, kterou bude používat výsledná aplikace. Nejefektivnější cesta vede přes tzv. XML feed soubory. Jedná se o speciální datový soubor ve formátu XML, který obsahuje informace o nabízených položkách (název, popis, cena, údaje o dostupnosti atd.) [8].

XML, anglicky EXtensible Markup Language, je značkovací jazyk podobný HTML. XML dokumenty mohou mít libovolnou strukturu a lze v nich používat libovolně pojmenované a strukturované prvky. V XML si tedy definujeme své vlastní tagy a udáváme tak strukturu dokumentu. Stejně tak i v našem případě budou mít XML soubory daný formát pro import dat [9].

Tyto XML feed soubory se poté používají pro tzv. srovnávače zboží a cen. Jedná se o webovou aplikaci, do které uživatel zadá produkt a která následně hledá napříč e-shopy. Výsledkem je seznam produktů z různých e-shopů včetně jejich ceny, dostupnosti a prodejních míst. Účelem těchto srovnávačů je poskytnout zákazníkovi informace o nejlepší ceně a dostupnosti hledaného produktu [10].

## 3.1 Specifikace XML souboru

Formát XML souboru pro import produktů do databáze má většina srovnávačů buď stejný, nebo velice podobný. Každý srovnávač cen má tedy i jednu celou sekci ve své nápovědě věnovanou právě správnému formátu XML feed souboru, aby došlo ke správnému importu produktů. Pro účely výsledné aplikace jsem si vybral srovnávač Heureka.cz.

### 3.1.1 Srovnávač cen Heureka

Heureka.cz je český interaktivní nákupní rádce, který nabízí široké spektrum různých možností usnadňujících internetové nakupování. Nabízí široké spektrum služeb, jako je například porovnávání cen produktů, vyhledávání různých parametrů produktů, zajištění recenze produktů, atd.

Přidávání nových e-shopů do databáze Heureka.cz je jednou ze služeb obchodům. Uživatel si založí profil, kde vyplní všechny důležité informace o svém obchodu. Mezi povinnými údaji je samozřejmě URL obchodu, ale také URL XML importu, tedy cesta k XML souboru s produkty daného e-shopu.

Pro účely výsledné aplikace tedy budeme předpokládat, že e-shop, který chceme transformovat do mobilní aplikace, bude mít záznam v databázi na Heureka.cz. Právě URL XML feed souboru využijeme k naimportování všech produktů e-shopu do naší interní databáze.

### 3.1.2 Formát XML souboru

Srovnávač cen Heureka.cz má vlastní specifikaci XML souboru, který je potřebný pro import produktů do interní databáze. Na stránkách popisující služby Heureka.cz je přesně specifikováno, jaký formát musí daný XML soubor mít. My tento formát využijeme pro sestavení tabulek v naší databázi a poté naimplementujeme na našem serveru funkci, která bude tento formát XML zpracovávat a vkládat záznamy do našich tabulek.

Nejdříve se ale podíváme na daný formát a popíšeme si nejdůležitější značky. Specifikace XML souboru na Heureka.cz obsahuje několik různých druhů značek pro popis produktu. Ne všechny ale jsou povinné.

XML soubor musí povinně obsahovat kořenový element <SHOP>. V něm se poté nacházejí elementy <SHOPITEM>, který reprezentuje právě jeden produkt. Každý produkt má pak další povinné atributy, jako například jednoznačný identifikátor, název produktu, popis produktu, cenu atd. Více informací o formátu XML souboru pro srovnávač cen Heureka.cz naleznete v [11].

Výsledný XML soubor s jedním produktem potom může vypadat například takto:

```
<?xml version="1.0" encoding="utf-8"?>
<SHOP>
  <SHOPITEM>
    <ITEM_ID>AB123</ITEM_ID>
    <PRODUCTNAME>Nokia 5800 XpressMusic</PRODUCTNAME>
    <PRODUCT>Nokia 5800 XpressMusic + pouzdro zdarma</PRODUCT>
    <DESCRIPTION>Klasický s plným dotykovým uživatelským
rozhraním</DESCRIPTION>
    <URL>http://obchod.cz/mobily/nokia-5800-xpressmusic</URL>
    <IMGURL>http://obchod.cz/mobily/nokia-5800-
xpressmusic/obrazek.jpg</IMGURL>
    <IMGURL_ALTERNATIVE>http://obchod.cz/mobily/nokia-5800-
xpressmusic/obrazek2.jpg</IMGURL_ALTERNATIVE>
    <PRICE_VAT>6000</PRICE_VAT>
    <MANUFACTURER>NOKIA</MANUFACTURER>
    <CATEGORYTEXT>Elektronika | Mobilní telefony</CATEGORYTEXT>
  </SHOPITEM>
</SHOP>
```

V tomto příkladu jsou zobrazeny jen některé základní značky. Produkt může obsahovat více značek a některé se mohou vyskytovat i víckrát.

# 4 Požadavky na systém

V předchozích kapitolách jsme si uvedli, co je to webová aplikace, mobilní aplikace (ať už webová nebo nativní) a co je to e-shop. Náš systém bude tvořen z webové aplikace s přístupem k databázi, která bude umístěna na serveru, a z mobilní aplikace s vlastní lokální databází. Tato lokální databáze ale bude stahovat data z databáze na serveru. O transformaci dat ve formátu XML, popsaném v kapitole 3, se bude starat funkce, která tato data uloží do databáze umístěné na serveru.

Mobilní aplikace bude nativní aplikace pro mobilní zařízení s operačním systémem Android. Bude obsahovat lokální databázi, která si bude data stahovat ze serverové databáze. Odtud poté budou data zobrazována uživateli. Aplikace bude fungovat jako mobilní e-shop, kde bude mít uživatel k dispozici košík, do kterého bude moci přidávat produkty z katalogu. Mobilní aplikace bude umět vyhledávat produkty a bude umožňovat ukládání osobních informací uživatele. Samotné vytvoření objednávky proběhne uložením do databáze na serveru a odesláním emailu prodejci i kupujícímu o vytvořené objednávce.

## 4.1 Případy užití

Při návrhu aplikace bylo nejdříve potřeba vytvořit případy užití. Diagramy případů užití (use cases) se v RUP[3] používají na zachycení požadavků a jsou součástí jazyka UML. UML je jednotný grafický jazyk pro specifikaci, vizualizaci, konstrukci, dokumentaci při OO analýze a návrhu a pro modelování organizace (business modelling) [13].

„Klíčovými aktivitami metodiky RUP ve fázi specifikace požadavků jsou nalezení případů užití a jejich účastníků a nalezení detailů vybraných případů užití. Jeden případ užití je chápán jako funkce, kterou systém vykonává jménem jednotlivých účastníků nebo v jejich prospěch. Každý případ užití má svůj název, jednoznačný identifikátor a specifikaci.“ [14]

Vytvořil jsem tedy diagram případu užití pro všechny případy, které můžou v používání aplikace nastat, a jak se systém zachová. Celý diagram je k dispozici v [Příloha 2] a jednotlivé detaily případů užití v [Příloha 3].

## 4.2 Návrh databáze

Jedna z prvních věcí, které byly potřeba navrhnout k tomu, aby systém správně fungoval, je databáze. Jak jsem se zmínil v předchozím odstavci, budou dohromady existovat dvě databáze. Jedna databáze bude na serveru, která bude zaručovat aktuálnost dat z XML feed souboru. O tuto aktualizaci se bude starat funkce uložená na serveru, která v daných intervalech zkontroluje aktuálnost XML feed souboru, a pokud byl aktualizován, zaktualizuje také databázi na serveru.

Druhá databáze je lokální a je uložena přímo v mobilním zařízení. O naplnění této databáze aktuálními daty se znovu stará funkce, tentokrát ale v mobilní aplikaci, jež zajistí stejná data, která jsou v databázi, na server.

### 4.2.1 Tabulka s nastavením aplikace

V databázi je kromě produktů a dalších dat pro fungování e-shopu uloženo také nastavení aplikace. Aby byla aplikace co nejméně závislá na informacích, které musel programátor napsat do zdrojového

[3] Metodika vývoje softwaru Rational Unified Software [12].

kódu, vybírá většinu nastavení právě z databáze. V této tabulce jsou uloženy informace o konkrétním e-shopu, jako je název obchodu, URL obchodu, kontaktní email, obchodní podmínky, URL loga e-shopu, URL XML feed, ze kterého se právě data o produktech stahují, částka, po jejímž přesažení je doprava zdarma, a také texty, které se zobrazí po úspěšné objednávce nebo po neúspěšné objednávce v aplikaci nebo v mailu. Také se zde ukládá datum poslední synchronizace se serverem. Záznam v této tabulce existuje jen jeden a je vždy aktualizován s novou synchronizací. Data zadává uživatel prostřednictvím webové aplikace popsané níže.

### 4.2.2 Tabulky pro produkt

Do této tabulky se ukládají data o jednom produktu. Jednotlivé sloupce vycházejí z dat, které nám poskytne XML feed soubor. Ovšem neobsahuje úplně všechna data, které může XML feed poskytnout, protože ten ještě navíc poskytuje dodatečné informace přímo pro server Heureka.cz a další. My se spokojíme se základními daty o produktu, aby si o něm mohl zákazník co nejvíce přečíst přímo z aplikace.

Základní informace, které potřebuje uživatel o produktu vědět, jsou: název, obrázek, popis, cena a jméno výrobce. XML feed nám všechny tyto informace poskytl. Můžeme z něj zjistit i EAN kód, popřípadě ISBN kód, pokud jde o knihu, typ zboží (zde se uvádí, jestli jde například o bazarové zboží), další příplatky (například balné), výrobce, kategorie nebo dodatečné obrázky. I tyto informace jsem zahrnul do tabulky pro produkt, které mohou nebo nemusí být vyplněné.

### 4.2.3 Tabulka pro typ dovozu

Při vytváření objednávky si musí nakupující vybrat typ dovozu svého zboží (osobní převzetí, PPL, atd.). Údaje o těchto typech dovozu jsou uloženy zvlášť v tabulce a jsou znovu zadávána uživatelem prostřednictvím webové aplikace.

Produkt má navíc specifikovanou kategorii. Kategorie mohou mít také stromovou strukturu, s čímž musíme také počítat. Byly tedy vytvořeny zvlášť tabulky pro kategorie i pro produkty a pro spojení těchto tabulek bylo využito cizích klíčů [15].

### 4.2.4 Tabulky pro objednávku

Tabulky, které uchovávají informace o uzavřených objednávkách, bylo potřeba navrhnout tak, aby i po pozdějším zobrazení dat pořád ukazovaly hodnoty, které byly platné při uzavření objednávky. Je tu hlavně myšlena cena produktu v době uzavření objednávky, která později při zobrazení těchto dat už nemusí odpovídat.

Samotná tabulka objednávky obsahuje sloupce pro uložení základních informací o uživateli, který si zboží objednal. Můžeme z ní tedy zjistit informace jako: jméno, příjmení, email, telefon, adresu, město, PSČ, IČ nebo DIČ. Samozřejmě nesmí chybět datum, kdy byla objednávka uzavřena, a informace o dodání (typ a cena).

K této tabulce dále existuje tabulka, která obsahuje záznamy odkazující na objednaný produkt a referenci na danou objednávku. Do této tabulky s objednaným zbožím se také zapisuje cena zakoupeného produktu, která je v době objednávky platná, vybraná velikost (pokud je to možné) a počet kusů.

Z tohoto návrhu poté vznikl celý ER Diagram, který je dostupný v příloze [Příloha 1].

### 4.2.5 Komunikace s databází v mobilní aplikaci

Mobilní aplikace bude komunikovat jak s databází na serveru, tak s lokální databází. Většina komunikace bude v mobilní aplikaci právě s lokální databází, protože komunikace s databází na serveru má jen zajistit aktuálnost dat v databázi lokální.

Komunikace s lokální databází bude zajištěna pomocí open-source databázového systému SQLite speciálně pro .NET. SQLite je malý, rychlý a lehce přenositelný. Databáze je uložena jen v jednom souboru. Je v něm implementována většina standardu SQL92 a formát databázového souboru je multiplatformní, takže je jednoduché ho kopírovat mezi 32-bit a 64-bit operačním systémem nebo mezi big-endian a little-endian architekturami.

Je oblíbeným databázovým systémem u paměťově omezených zařízeních, jako například mobilní zařízení, PDA nebo MP3 přehrávače.

Protože je SQLite navrženo tak, aby bylo co nejmenší a nejrychlejší, má také pár omezení. Je potřeba se rozhodnout, kam také databázový soubor uložit. U Androidu je možné se rozhodnout mezi interním a externím uložištěm.

Nelze použít jedno SQLite připojení napříč vlákny. Proto je potřeba si dávat pozor na to, aby otevření a uzavření připojení proběhlo vždy v jednom vláknu. Je možné tomu také předejít pomocí zámků, ale tam zase vzniká nebezpečí deadlocku [16].

Transakce SQLite jsou ACID i když jsou přerušeny z důvodu pádu systému nebo ztráty energie. ACID je zkratka pro základní vlastnosti transakce. Zkratka je tvořena ze začátečních písmen anglických názvů vlastností. Jsou to tyto vlastnosti:

- atomičnost (Atomicity)
- konzistence (Consistency)
- izolace (Isolation)
- trvalost (Durability)

**Atomičnost** znamená, že transakce přestavuje z hlediska provedení jeden celek. Buď je tedy provedena celá transakce, nebo se neprovede nic.

**Konzistence** znamená, že izolovaná transakce (není ovlivňována žádnými jinými souběžně probíhajícími transakcemi) neporušuje konzistenci databáze. Platí tedy, že pokud byla databáze v konzistentním stavu před zahájením transakce, musí být v konzistentním stavu i po skončení transakce.

**Izolace** znamená, že pokud bude probíhat několik transakcí souběžně, bude zajištěno, že transakce budou z hlediska provádění operací s daty v databázi izolované. Tedy efekt jejich operací bude takový, jako kdyby neprobíhaly souběžně, ale jedna za druhou.

**Trvalost** znamená, že po úspěšném dokončení transakce budou mít všechny změny v databázi trvalý charakter, a to i při výpadku systému [15].

Dotazy nad SQLite databází lze provádět jak pomocí LINQ (popsáno v kapitole 6) nebo použít dotaz v jazyce SQL.

## 4.3 Implementace serverové části systému

Serverová část systému se stará o transformaci XML souboru do interní databáze a poskytuje API pro komunikaci s mobilní aplikací. Server je implementován jako webová aplikace napsaná v ASP

.NET[4]. Obsahuje jen jednoduchou administrační část pro zadávání základních informací o obchodě, které jsme si popsali výše, a uživatel také může přidávat nebo mazat druhy dovozu zboží pro svůj obchod, jež pak může uživatel v aplikaci vybrat.

Aplikace obsahuje sadu metod pro transformaci serializovaných dat do databáze. Databázové tabulky jsou totožné jako v mobilní aplikaci, aby spolu tyto dvě aplikace mohly přímo komunikovat bez jakéhokoliv upravování. Při vyvolání požadavku o synchronizaci produktů si aplikace nejdřív zjistí, kdy byl XML feed soubor naposledy upraven. To zjistí z HTTP hlavičky „Last-Modified". Toto datum porovná s datem poslední synchronizace, kterou si uchovává v databázi. Pokud byl XML soubor změněn po poslední synchronizaci, provede se nová synchronizace. Na stejném principu pracuje i synchronizace v mobilní aplikaci, která porovnává datum poslední synchronizace v databázi na serveru a v lokální databázi v mobilní aplikaci.

Server také implementuje metodu, kdy po obdržení a uložení objednávky do databáze odešle na email objednávajícího a provozovatele e-shopu informační email s detailem objednávky. Tento email slouží hlavně pro potvrzení odeslání objednávky objednávajícího a pro informaci o nové objednávce provozovateli.

Poslední částí serverové části systému je Web API. Využil jsem k tomu framework ASP .NET Web API. Tyto API jsou postavené na architektuře RESTful, a jsou dostupné prostřednictvím standardních HTTP metod (GET, POST, PUT, DELETE) [18].

Mobilní aplikace se serveru na data dotazuje prostřednictvím http klienta. Data jsou přenášena ve formátu JSON, a poté si je mobilní aplikace sestaví zpátky do objektů pomocí příslušného převodníku. O tom ale později v kapitole 6.

V tento moment tedy máme implementovanou serverovou část systému, jenž nám na vyžádání dokáže transformovat data z XML souboru do databáze a dokáže nám tato data poslat ve formátu JSON. Obsahuje také metody využívající HTTP metodu POST pro ukládání dat do databáze. Implementace metod využívajících PUT a DELETE nebylo pro tento systém potřeba.

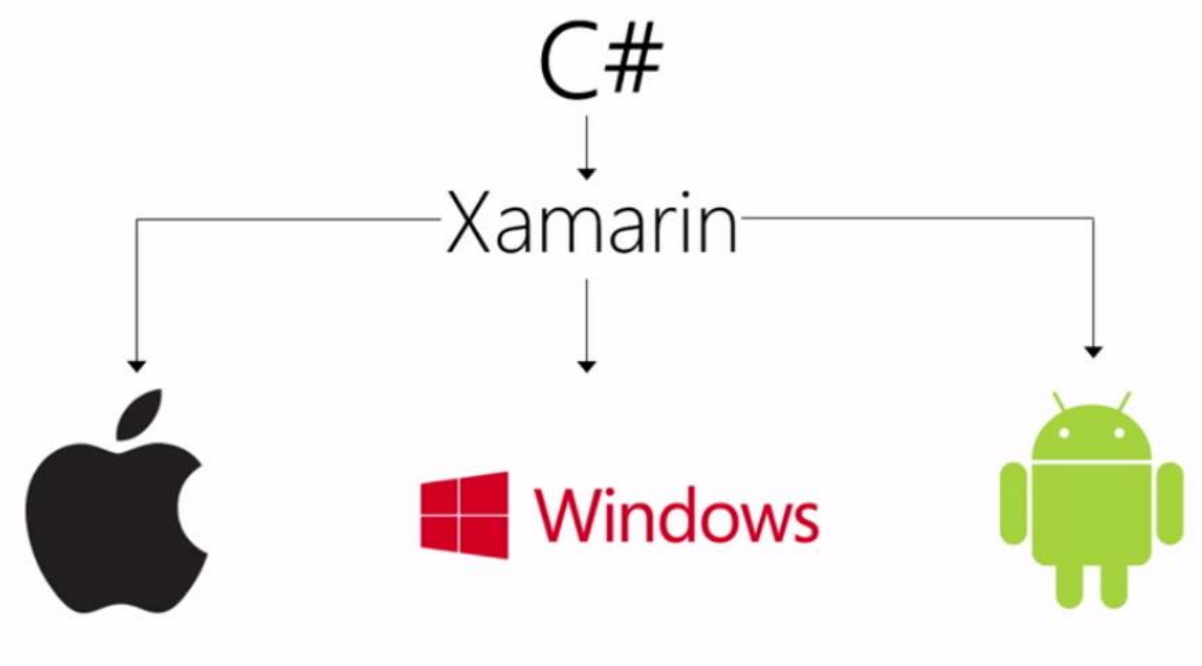
[4] Webový framework pro vytváření webových stránek a aplikací s použitím HTML, CSS a Javascriptu. Slouží také k vytváření Web API, mobilních stránek atd. Programuje se v jazyce C# [32].

# 5 Platforma Xamarin

Firma Xamarin, dříve Ximian, byla založena v roce 2011. Jako firma Ximian vytvořila projekt Mono, v jehož vývoji i později jako Xamarin pokračovala. První tři roky existence jako Xamarin se nejvíce zaměřovala na tři základní .NET knihovny:

- Xamarin.Mac, také známá jako MonoMac.
- Xamarin.iOS, také známá jako MonoTouch.
- XamarinAndroid, také známá jako Mono for Android nebo také MonoDroid.

Tyto knihovny jsou společně známé právě jako Xamarin Platforma. Knihovny obsahují nejen .NET verze nativních Mac, iOS a Android API, ale také přístup do .NET Framework knihoven.
Xamarin vývojářům uvolnil nástroj Xamarin Studio, což je integrované vývojové prostředí, které běží jak na PC, tak na Mac. Xamarin také umožňuje strukturovat kód pro sdílení mezi více aplikacemi [19].




*Obrázek: 5.1. Platforma Xamarin [20]*

## 5.1 O platformě Android

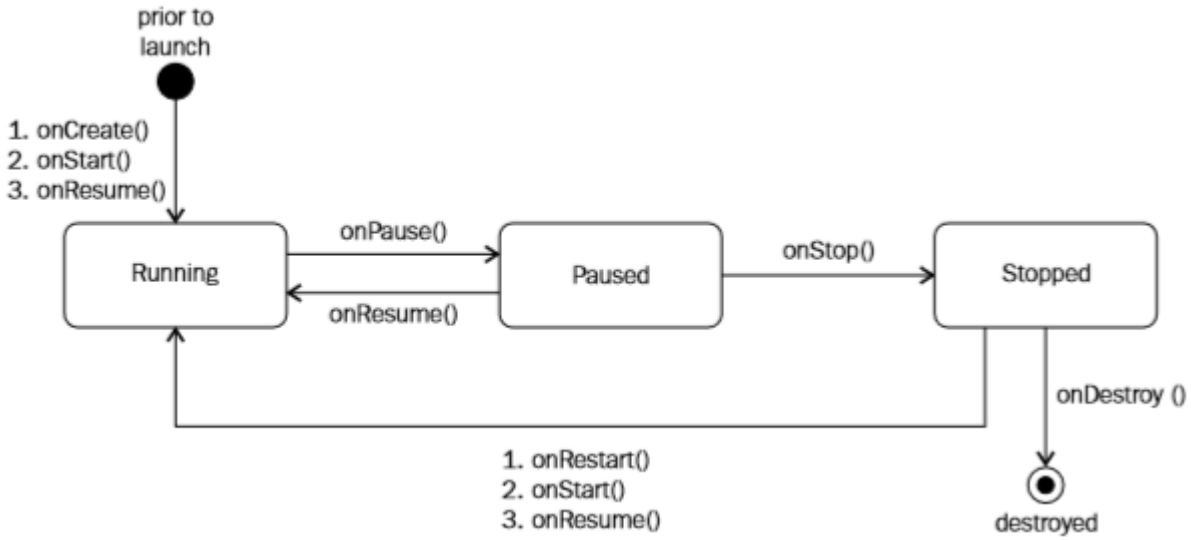
Přestože, je tento projekt programován v C# na platformě Xamarin, je také důležité vědět, jak funguje obyčejná aplikace pro Android napsaná v jazyce Java.

Android je operační systém založený na jádře Linuxu primárně pro mobilní zařízení.
Aplikace na Android běží pod Dalvik Virtual Machine (Dalvik VM), který je podobný Java VM[5], ale byl optimalizován pro zařízení s omezenou paměťovou a procesorovou kapacitou.
Aplikace na Android jsou kompilovány do Java Byte code za použití Java kompilátoru, ale mají ještě přidaný kompilační krok, který převede Java byte code do Dalvik byte code pro běh na Dalvik VM.

[5] Java Virtual Machine

Jedna z nejzákladnějších částí aplikace pro Android je tzv. aktivita. Aktivita poskytuje vždy jednu funkci, kterou může uživatel s aplikací provádět, např. vypsat si kontakty, přidat nový kontakt nebo zobrazit lokaci na mapě. Jedna aplikace je složená z několika aktivit.

Uživatel komunikuje s aktivitou pomocí jednoho nebo více View, o kterých si budeme vykládat později v rámci Xamarin.Forms. V návrhovém vzoru Model-View-Controller zastupuje aktivita právě controller [21].




*Obrázek 5.2: Životní cyklus aktivity [21]*

## 5.2 Architektura Xamarin.Android

Dříve, než se pustíme do samotného popisu architektury Xamarin.Android, měl bych pohovořit o tom, proč jsem si vybral právě tuto architekturu a jaké má výhody naproti psaní aplikace v jazyce Java.

Jako první věc, která mě zaujala na celé platformě Xamarin je fakt, že můžete většinu kódu sdílet mezi platformami. Právě díky projektu Mono, o němž se pobavíme později, mohu napsat většinu logiky společnou jak pro Windows Phone, Android nebo iOS. Z toho už je jasné, že pro všechny platformy se píše ve stejném jazyce a to je jazyk C#.

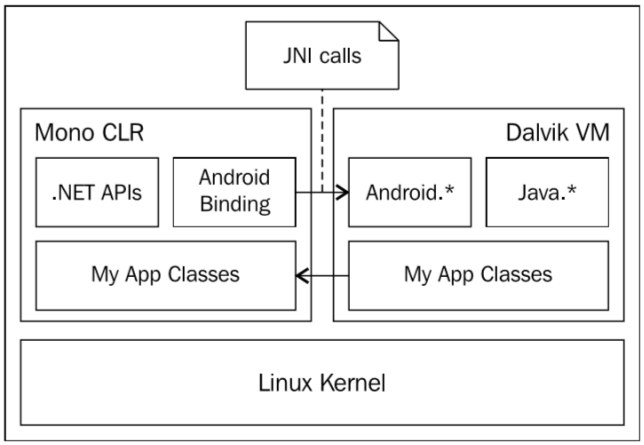
Tím se dostáváme k druhému důvodu, proč jsem si Xamarin vybral. Je jím jazyk C#, se kterým jsem přišel do styku jako s prvním vyšším[6] programovacím jazykem. I když jsem si poté vyzkoušel další jazyky jako je C, C++ nebo Java, zůstává pro mě C# stále nejoblíbenějším jazykem. Jeho snad jediná nevýhoda byla v tom, že byla pevně vázána na platformu .NET, tudíž aplikace napsané v tomto jazyce nemohly být dále přenositelné na jiné platformy. To se ale změnilo díky projektu Mono, který popíšu záhy a dokonce 12. listopadu 2014 Microsoft uvolnil .NET pod open-source licencí jako .NET Core [23]. Tímto se tedy otevírají dveře platformě .NET a tím i jazyku C# nové možnosti pro psaní multiplatformních aplikací.

[6] Vyšší programovací jazyk je v informatice označení pro programovací jazyk s vyšší mírou abstrakce [22].

## 5.3 Mono a Dalvik

O projektu Mono jsem se již zmínil v předchozí podkapitole. Mono je open-source multiplatformní implementace C# kompilátoru a Common Language Runtime (CLR), které je binárně kompatibilní s Microsoft .NET platformou. Mono CLR bylo portováno na několik platforem, včetně Androidu, většiny distribucí Linuxu, BSD, OS X, Windows, Solaris, Wii, Xbox 360, iOS nebo Playstation 3.

Dále jsem již psal něco málo o Dalvik VM. Víme, že je to virtuální stroj, na kterém běží aplikace psaná pro Android. Pokud se ale rozhodneme, že budeme psát aplikaci pro Android pod architekturou Xamarin.Android, musí souběžně běžet jak Mono, tak Dalvik.




*Obrázek 5.3: Komunikace mezi Mono CLR a Dalvik VM [21]*

Jak je poznat z obrázku, nad celou komunikací mezi Mono a Dalvik stojí ještě JNI volání. JNI (Java Native Interface) je framework, který umožňuje kódu nenapsaném v jazyce Java, jako třeba C++ nebo C#, aby volal, nebo byl volán kódem napsaném v jazyce Java běžícím na JVM. JNI je tedy hlavním komponentem v celém fungování architektury Xamarin.Android.

## 5.4 Xamarin.Forms

Xamarin.Forms je multiplatformní UI nástrojová abstrakce, jenž umožňuje uživateli vytvořit uživatelské rozhraní, které může být sdílené napříč platformami Android, iOS a Windows Phone.
Na základě cílové platformy Xamarin.Forms.Core použije dané Xamarin.Forms.Platform knihovny. Tyto knihovny jsou většinou sbírka tříd, které se nazývají renderery, transformující objekty Xamarin.Forms uživatelského rozhraní do uživatelského rozhraní specifické platformy.

V Xamarin.Forms se objekty, které uživatel vidí na obrazovce, nazývají „visual elements“ a mají 3 hlavní kategorie:

- page
- layout
- view

Máme více druhů Stránek (Pages). Na těchto Stránkách jsou vizuální elementy poskládány hierarchicky (rodič, potomek). Elementy mohou mít jednoho nebo více potomků. Termín view v Xamarin.Forms představuje známé typy prezentačních a interaktivních objektů, jako jsou tlačítka, přepínače, progress bary nebo date pickery. Většinou se jim říká „controls“ nebo také „widgets“.

Pro vytvoření uživatelského rozhraní v Xamarin.Forms jsou dva způsoby. První způsob je vytvoření UI čistě ve zdrojovém kódu s využitím bohatého API, které knihovna nabízí. Druhá možnost je použití Extensible Application Markup Language (XAML). Jde o jazyk vytvořený firmou

Microsoft na bázi XML pro inicializaci a instanciaci objektů. Samotné uživatelské rozhraní je definováno v XML souboru za použití XAML syntaxe. Já jsem se rozhodl všechny UI prvky napsat v jazyce XAML.

Xamarin.Forms rozlišuje několik rozložení stránek, jichž je několik obsahují už danou strukturu pro správné rozložení elementů, např. Content Page, Master DetailPage, NavigationPage atd [24]. Dále jsou tu komponenty Layouts, které se chovají jako kontejnery pro další Layouty nebo Views. Typicky obsahují logiku pro určení pozice a velikosti jejich potomků. Takovými Layouty mohou být např. StackLayout, GridLayout nebo ScrollView [25].

# 6 Tvorba mobilní aplikace

Dosud jsme si procházeli všechny důležité nástroje a informace, které budeme potřebovat pro samotnou tvorbu našeho systému. Nejdříve začneme s tvorbou mobilní aplikace jen s lokální databází, abychom jí poté mohli umožnit komunikaci s databází na serveru.

Tato mobilní aplikace bude primárně psána pro operační systém Android, nicméně bude mít i část kódu přenositelnou mezi všemi platformami (iOS, Android, Windows Phone) stejně tak UI bude napsáno pomocí Xamarin.Forms, které zajistí přenositelné UI.

Důvodem, proč jsem nevytvořil aplikace pro všechny 3 platformy je studentská licence Xamarin. Tato licence mi dovoluje vytvářet jen pro platformy Android a iOS. K Windows Phone projektu bych musel mít vyšší licenci, která dovoluje využívání Visual Studia s Xamarin projekty. Pro iOS tvorbu aplikací je zase potřeba tento projekt vytvořit a přeložit na zařízení s operačním systémem OS X. V tomto případě jsem byl tedy schopen vytvořit projekt jen pro mobilní zařízení se systémem Android.

Pro psaní zdrojových kódů jsem využíval integrované prostředí Xamarin Studio. Jde o upravenou verzi Mono Develop IDE, která může být použita pro vytváření Android, iOS nebo OS X aplikací. Xamarin Studio je dostupné jak ve verzi pro Windows, tak pro OS X [21].

Tato aplikace je složená celkem ze dvou projektů. Prvním projektem je tzv. Shared Project (sdílený projekt), na který mohou odkazovat další projekty v jednom řešení (solution). Kód v tomto projektu je kompilován jako část každého projektu, jenž na něj ukazuje a může obsahovat kompilační direktivy pro správnou funkci na specifické platformě [26]. Část kódu je tedy uložená v tomto sdíleném projektu. Dále v řešení existuje projekt pro platformu Android. Toto řešení je tedy již nachystané k případnému rozšíření na další platformy.

## 6.1 Architektura MVVM

Ještě před popisem struktury našeho programu si řekněme něco o architektuře, podle které budeme celou aplikaci tvořit. Architektura MVVM se hojně využívá právě pro tvorbu klientských aplikací na desktop a na mobilní platformy.

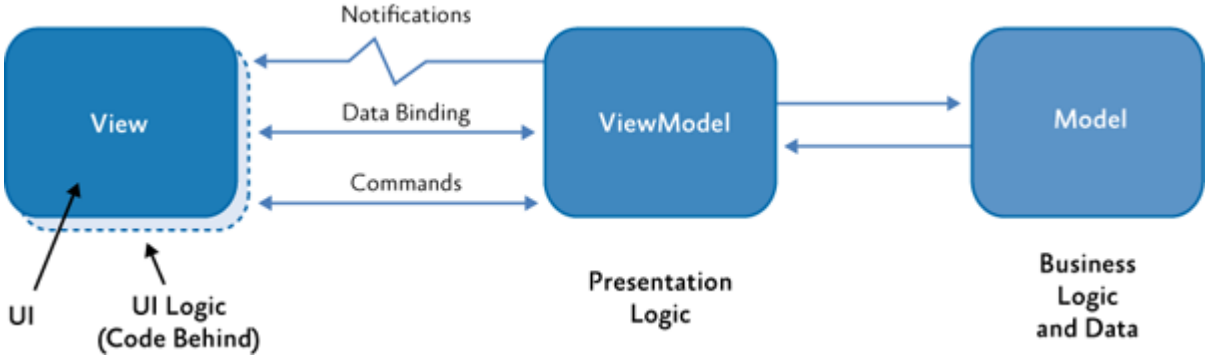
Zkratka architektury MVVM je složená z počátečních písmen anglických slov Model, View a ViewModel. Tato architektura se objevila jako odpověď na některá omezení architektury MVC a zároveň využívá její silné složky. V této architektuře byla logika a stav zobrazovaných dat přesunuta do externí složky, která dostala název ViewModel. Tento ViewModel má na starosti udržování stavu View a vykonávání View logiky. Díky ViewModelu byl představen koncept externích vlastností (property) objektů.

Tyto vlastnosti poskytují aktualizace stavu objektů při změnách. Této myšlence se říká two-way data binding, neboli dvoucestné plnění dat. Vlastnosti umožňují změnu objektů z View. Robustní podpora této funkcionality přišla s .NET 3.0 při uvedení Silverlight pro vytváření klientů a Windows Presentation Foundation (WPF). Struktura MVVM je dosti podobná MVC nebo MVP.

**Model** je reprezentace dat v paměti převzatá z perzistentního uložiště. Model je také zodpovědný za oznamování view modelu o změnách.

**View** je i pod MVVM zodpovědné za zobrazování dat, sbírání vstupních dat a jejich předávání jen s tím rozdílem, že teď jsou předávána do ViewModel. View využívá tzv. binding systém (naplňování dat) pro komunikaci s View modelem, k čemuž už nepotřebuje žádný zvláštní kód přímo ve View (code behind).

**ViewModel** nebo presentation model je zodpovědný za stav View a jeho logiku. Spoléhá také na binding systém pro komunikaci s View. ViewModel je takový prostředník v architektuře MVVM a stará se také o přesouvání dat z Modelu do View a naopak. Jeho hlavní role je, jak už jeho název napovídá, „zobrazení modelu“ ve formě, která je lépe zobrazitelná jako stav View. View model je také zodpovědný za implementování View logiky pro transformaci stavu View podle potřeby [27].




*Obrázek 6.1: Architektura MVVM [28]*

# 6.2 Rozdělení aplikace

Pro zavedení návrhového vzoru Model-View-ViewModel do naší aplikace je potřeba si ji rozdělit na logické celky:

- SQLite Records
- Data Access Layer
- Model Classes
- Data Service Layer
- ViewModels

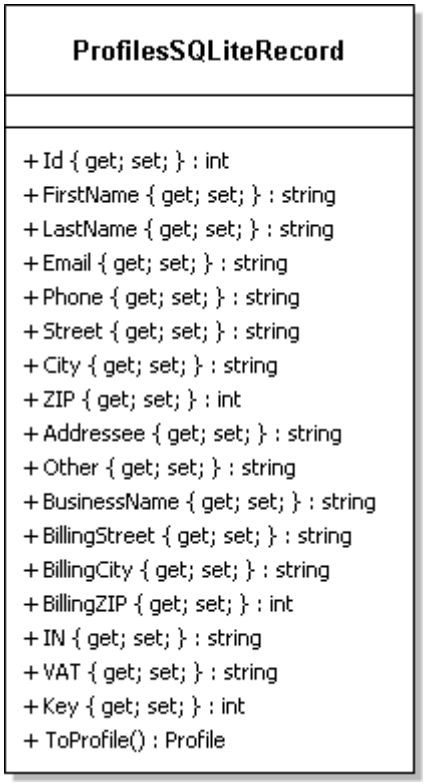
Diagramy jsou k dispozici v příloze. [Příloha 4], [Příloha 5], [Příloha 6], [Příloha 7] a [Příloha 8].

# 6.3 Data Access Layer

Nejnižší logický prvek jsem nazval Data Access Layer, tedy vrstvu s přímým přístupem k datům. Jde o sadu tříd pracující přímo s databází. Sestává ze tříd, definujících strukturu tabulek v databázi a ze tříd pro každou tabulku implementujících operace nad databází (čtení, zápis).

## 6.3.1 SQLite Record třídy

Třídy SQLite Record slouží jako předpisy struktury databázových tabulek, které mi SQLite .NET na jejich základě vytvoří. Samotné neobsahují žádnou logiku. Stejná struktura databáze bude i na serveru, jen s tím rozdílem, že lokální si bude ještě navíc uchovávat profily uživatelů.




*Obrázek 6.2: Třída ProfileSQLiteRecord pro vytvoření tabulky v databázi*

### 6.3.2 Data Access Layer třídy

Pro každou ze tříd SQLite Record existuje korespondující Data Access Layer třída, která už aplikuje nějakou logiku. Typicky jde o ukládání a výběr dat z databáze nebo do databáze. Při výběru dat jsou tato data vrácena ve složitější struktuře. K tomu slouží modelové třídy, které sestávají ze složitějších atributů a které poté slouží k zobrazování dat uživateli.

## 6.4 Data Service Layer

Tato vrstva poskytuje data do ViewModel, která budou zobrazena uživateli. Používají Data Access Layer třídy, díky nimž mohou přistupovat do databáze. Tato data si cachují (ukládají do vyrovnávací paměti), aby je poté mohly rychle poskytnout do ViewModel. Pro výběr dat jak z databáze u Data Access Layer i v Data Service Layer používám sadu dotazů jazyka LINQ, který si popíšeme v další podkapitole.

### 6.4.1 LINQ

LINQ (Language Integrated Query) je sada jazykových a frameworkových nástrojů pro psaní strukturovaných typově bezpečných dotazů nad lokálními objektovými kolekcemi nebo vzdálenými zdroji dat.

LINQ dovoluje vykonat dotaz nad všemi kolekcemi implementující rozhraní IEnumerable<T>, stejně tak dokáže vykonávat dotazy nad tabulkami na SQL serveru. V Enumerable třídě v System.Linq je okolo 40 dotazovacích operátorů, kde jsou všechny implementovány jako statické rozšiřovací metody. Nazývají se standardní dotazovací operátory. Většina dotazovacích operátorů podporuje argument zapsaný pomocí lambda výrazu. Lambda výraz

pomáhá upravovat a zjednodušovat a upravovat dotaz. Pro příklad lambda výraz pro délku větší, rovno 4 je následující:

```
(n => n.Lenght >= 4
```

Nejčastěji se argumenty zapsané pomocí lambda výrazu používají u operátoru Where nebo Select. Celkový LINQ dotaz může pak vypadat následovně:

```
IEnumerable<string> filteredNames = names
.Where(n => n.Contains(‚a‘))
.Select(n => n.ToUpper());
```

Tento ukázkový kód vybere ze seznamu `names` všechny záznamy, které obsahují písmeno ‚a‘, a vrátí ta jména s velkými písmeny. Byl také zapsán v tzv. Fluent syntax. Tato syntaxe je velice flexibilní, jednoduchá a intuitivní. Dále existuje také Query syntax, která je více podobná syntaxi dotazu v jazyce SQL a vždy začíná klauzulí `from` [29].

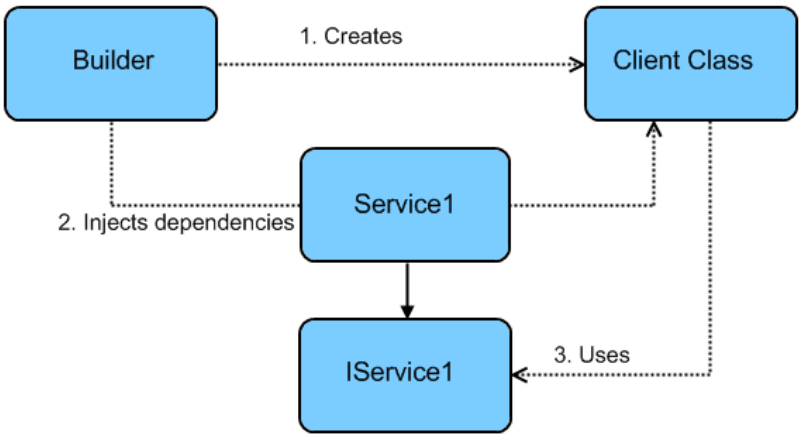
Stejného výsledku jako v předchozím příkladu zapsaném ve Fluent syntax se dá dosáhnout i pomocí Query syntax takto:

```
IEnumerable<string> filteredNames = from n in names
where n.Contains(‚a‘)
select n.ToUpper();
```

LINQ tedy nabízí velice mocný nástroj na dotazování nad kolekcemi, lze jím dosáhnout stejných výsledků jako pomocí jazyku SQL, a ještě mnohem více.

### 6.4.2 Dependency Injection

Pro vytváření instancí tříd Services DataAccessLayer a dalších využívám techniku Dependency Injection (DI), také známou jako Inversion of Control (IoC). Pomocí DI se mohu zbavit závislostí u tříd na získávání objektů, které potřebují ke své činnosti. Automatické vytvoření této instance mi zajišťuje právě technika Dependency Injection za použití Dependency Injectoru, v mém případě je to konkrétně Ninject, o kterém si nastíníme více v další podkapitole.




*Obrázek 6.4: Ukázka techniky Dependency Injection [30]*

### 6.4.3 Ninject

Ninject je open source dependency injector pro platformu .NET. Já jsem v mém projektu použil jeho verzi pro přenositelné knihovny (Ninject for Portable Libraries), který funguje pro Windows 8, Windows Phone 8, iOS nebo Android. Kromě tohoto injectoru existuje pro platformu .NET i mnoho dalších, ale já si tento vybral pro jeho jednoduchost, elegantnost a minimum konfigurace.

Pro používání Ninject je potřeba nejdříve vytvořit instanci Ninject kernelu (jádro). Tento objekt budeme používat ke komunikaci s Ninject. [31]

V mém případě mám instanci Ninject kernelu vytvořenou ve třídě MainActivity, kde je popsáno chování při spuštění aplikace. V této třídě mám statickou vlastnost (property) pro Ninject kernel, jehož instanci naplním při spuštění aplikace.
Vytváření závislostí, neboli Binding v Ninject se řídí následující instancí:

```
InstanceKernelu.Bind<IInterface>().To<Trida>();
```

Zde `Trida` představuje třídu, která implementuje rozhraní `IInterface`. Poté tam, kde potřebujeme vytvořit instanci třídy `Trida`, zavolám moji instanci jádra a vyžádám si od něj naplněnou instanci:

```
InstanceKernelu.Get<IInterface>();
```

Právě pro případ, abych se k instanci Ninject kernelu dostal odkudkoliv z aplikace, vytvořil jsem si zmíněnou statickou vlastnost, která se naplní vždy jen ve třídě MainActivity.

Ninject má také další zajímavou vlastnost a to je vytváření řetězce závislostí. Umí vyřešit a vytvořit instance všech tříd, které jsou potřebné pro vytvoření konkrétní instance. Ninject celkově nabízí velkou škálu možností vytváření závislostí. Umožňuje nastavit hodnoty vlastností při vytváření instance, nebo podporuje podmíněné Bindování. Více o Ninject na oficiálních stránkách [32].

## 6.5 ViewModels

Třídy ViewModels si uchovávají stav View a aplikují jeho logiku. Sestává tedy z atributů, které jsou nabindovány ve View. Některé atributy jsou nabindovány i dvoucestně, aby mohl uživatel stav těchto atributů měnit (například při zadávání osobních údajů, atd.).

ViewModels také obsahují tzv. Commands, což jsou objekty, které jsou navázány na nějaký kontrolní prvek ve View (tlačítko, přepínač, …). Implementují rozhraní ICommand a jsou spojeny s určitou funkcí, která mění stav View model. Commands také vědí, jestli mohou být spuštěny nebo ne. K tomu slouží funkce `CanExecute()`, jež každý objekt implementující rozhraní ICommand obsahuje a která vrací booleovskou hodnotu o tom, jestli může být daný Command vykonán nebo ne. Pomocí funkce také mohu daný Command zakázat, což má za následek automatické zakázání prvku ve View, na nějž je Command navázán [33].

## 6.6 Views

V mém projektu jsem Views psal pomocí jazyka XAML. Společně s XAML souborem je vytvořen také CS soubor s kódem v C# (code behind), kde mám přístup ke všem komponentám, které si nadefinuji v jazyce XAML. Mohu tedy design psát na dvou místech najednou. C# kód mi dává možnost použít standardní programovací techniky, což mi jazyk XAML neumožňuje. Díky C# kódu je pak například zkonstruován strom kategorií ve výsledné aplikaci. V tomto kódu také definuji tzv. binding context pro celé View. Jde o objekt, ze kterého bude bindovat atributy, v mém případě to jsou třídy ViewModels. Poté v jazyce XAML klíčovým slovem Binding přímo odkazuji na atributy nebo Commands v daném ViewModel.

## 6.7 Synchronizace se serverem

Hlavní fungování celého systému závisí na správné komunikaci aplikace se serverem. Mobilní aplikace ke své činnosti potřebuje připojení k internetu a dostupnost serveru. Obě tyto vlastnosti si sama aplikace při svém startu ověří, a pokud nebude k dispozici jedna z těchto možností, nepustí uživatele dál.

Pokud ale aplikace má přístup k internetu a server je dostupný, přikročí se k samotné synchronizaci. Pokud se jedná o úplně první spuštění aplikace na mobilním zařízení, vytvoří se databázový soubor s tabulkami a započne stahování veškerých dat ze serveru. Fakt, jestli se jedná o první spuštění, si aplikace zjistí prostým ověřením existence databázového souboru v paměti telefonu. Při dalších spuštěních už jen porovnává poslední datum synchronizace v lokální databázi oproti datu ze serveru. Pokud je datum na serveru novější než datum z lokální databáze, započne synchronizace. Synchronizují se tabulky s produkty, kategoriemi, nastavením a s typy dovozů. Pokud je ale datum v lokální databázi novější než v databázi na serveru, použije aplikace data, která má k dispozici v lokální databázi z poslední synchronizace.

Dále jsou také synchronizovány objednávky, a to přímo na serveru. Po dokončení objednávky na mobilním zařízení jsou prostřednictvím HTTP protokolu metodou POST zaslány informace o objednávce na server, který si tato data uloží do databáze. Jakmile je objednávka uložena v databázi, server odešle email s přehledem objednávky jak zákazníkovi, jenž si objednávku objednal, tak na kontaktní email, který provozovatel obchodu zadal na webové aplikaci na serveru.

# 7 Popis fungování systému

Koncept celého systému je postaven na myšlence univerzální e-shopové mobilní aplikace, která si konkrétní data o produktech stáhne ze serveru. Systém se tedy, jak už bylo řečeno, skládá ze serverové části a klientské části. V této kapitole si popíšeme fungování každé části zvlášť. Webová aplikace je uložená na cloudové platformě Microsoft Azure, kde také využívá plánovač pro opakované spuštění synchronizace.

## 7.1 Zvolený vzorek dat

Abych mohl plně popsat fungování systému, je potřeba si ho vyzkoušet na vhodně zvoleném vzorku dat. V následujících podkapitolách tedy bude systém ukázán pro e-shop PruhovanySvet.cz, jehož XML feed mi byl poskytnut. PruhovanySvet.cz je středně velký e-shop s oblečením čítající okolo 200 produktů rozdělených do příslušných kategorií.

## 7.2 Nastavení aplikace na serveru

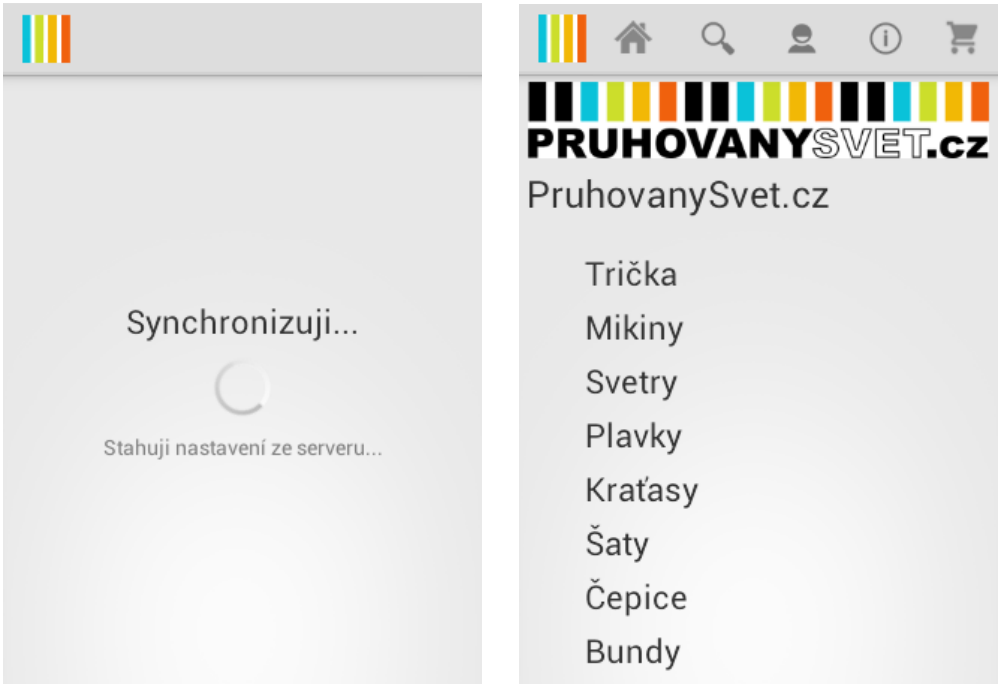
Ze všeho nejdřív musí uživatel vyplnit platná data prostřednictvím webové aplikace, která je uložená na serveru. Tato data jsou poté uložena do tabulky s nastavením aplikace, jež jsme si popsali v kapitole 4 při návrhu databáze. Jakmile se tato data uloží, bude server schopen zpracovat a uložit data z XML feed, který uživatel zadal. Spuštění této akce záleží na nastavení plánovače úloh, který tuto akci bude spouštět. Jakmile se spustí synchronizační akce a jsou správně zadaná data, zpracují se data z XML feed a uloží se do databáze na serveru.


*Obrázek 7.1: Ukázka webové aplikace pro zadání informací o e-shopu*

## 7.3 Ukázka mobilní aplikace

Aby mohla mobilní aplikace správně fungovat, je přece jenom nezbytné, aby se do zdrojového kódu zasáhlo. Je zapotřebí přímo napsat název aplikace, který se bude zobrazovat na konkrétním mobilním zařízení. Další povinnou věcí je napsat URL serveru, jenž poskytuje API pro komunikaci. Jako poslední je nadefinování vlastní ikony aplikace. Tato část může být brána jako nepovinná, jinak se zobrazí přednastavená ikona.

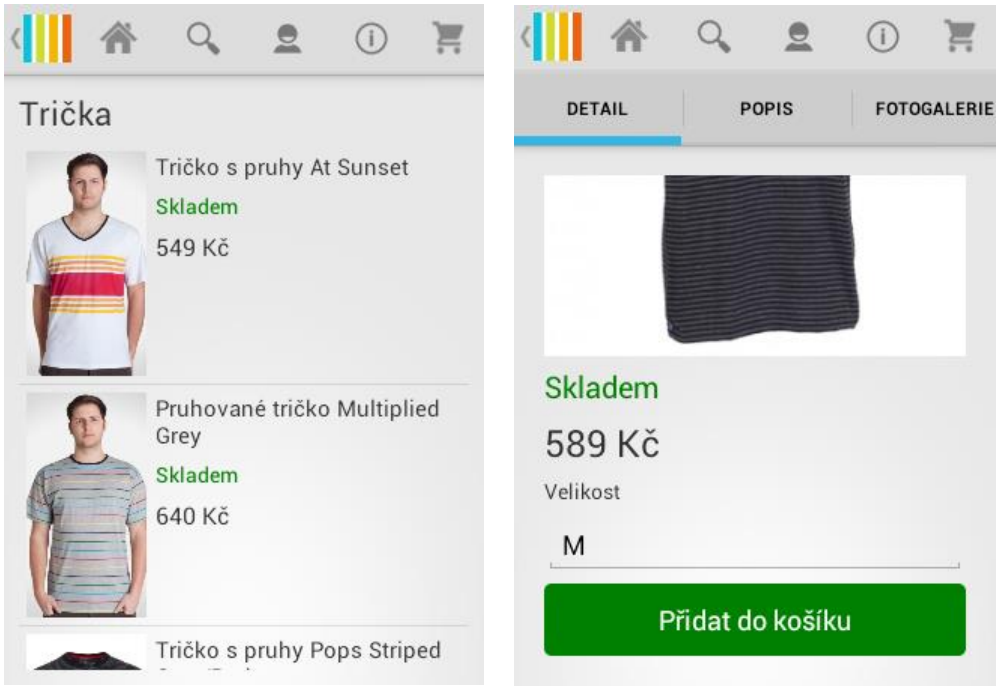
Dále už není potřeba zasahovat do kódu. Všechny ostatní informace již aplikace stahuje přímo ze serveru.




*Obrázek 7.2: Ukázka synchronizace při spuštění a následně hlavní strana s kategoriemi*

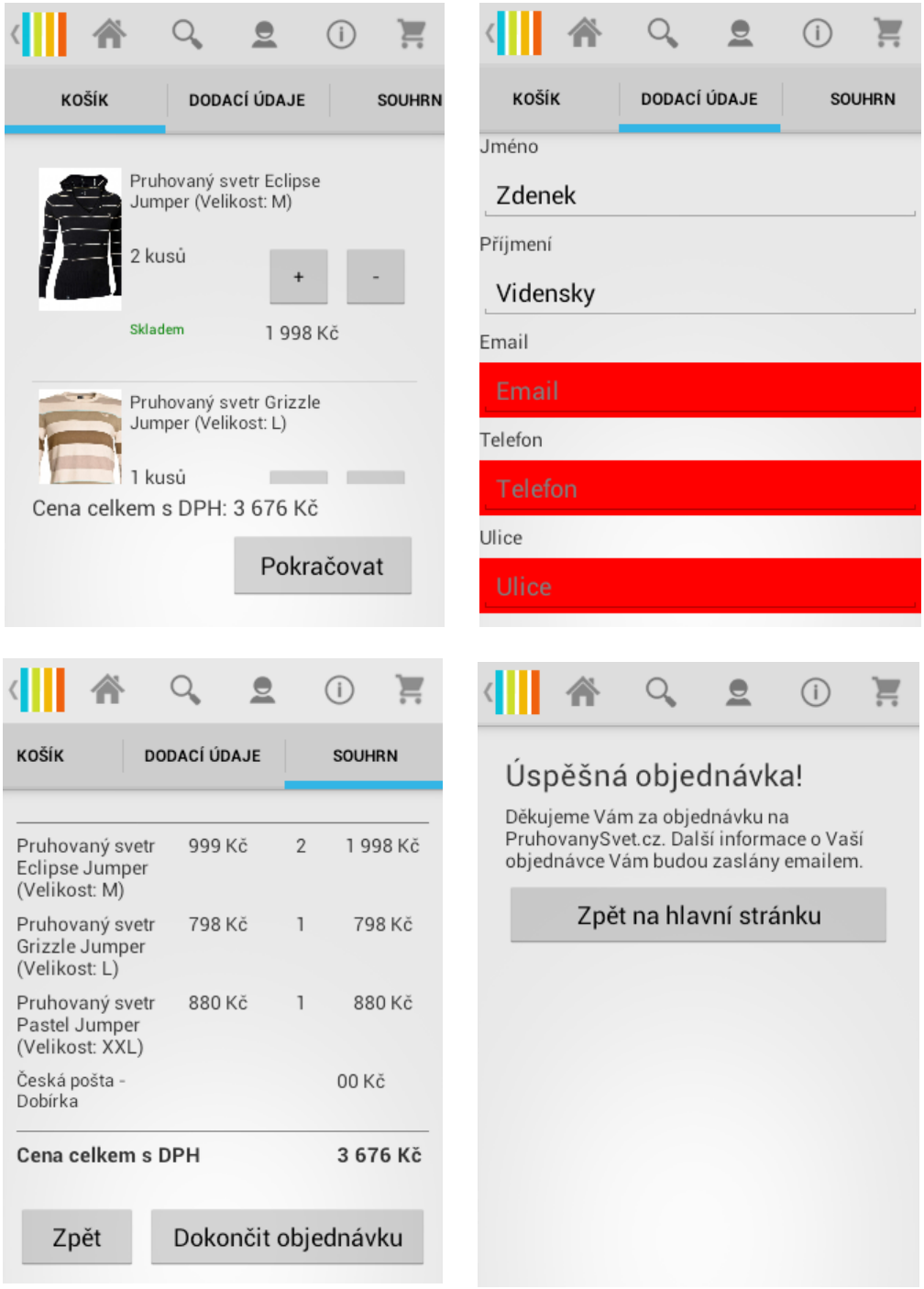
Po startu aplikace a její úspěšné synchronizaci se serverem se jako hlavní obrazovka zobrazí seznam kategorií s logem a názvem e-shopu. Při kliknutí na název kategorie se zobrazí seznam produktů patřících do této kategorie, případně produkty patřící do dalších podkategorií. V horní části u verze pro Android jsou k vidění navigační ikony pro (zleva) domů, vyhledávání, úpravu profilu, zobrazení informací a zobrazení nákupního košíku.

V seznamu produktů si mohu zobrazit o každém produktu detail, který sestává ze základních informací, jako je název, jméno výrobce, cena, jestli je skladem nebo jeho detailní popis a popřípadě další obrázky. Stránka s detailem produktu obsahuje tlačítko pro vložení produktu do košíku, jehož obsah si lze zobrazit tlačítkem v horním panelu úplně vpravo.




*Obrázek 7.3: Ukázka seznamu zboží a detail konkrétního zboží*

Na stránce s obsahem košíku můžeme také přidávat nebo odebírat počet kusů našeho přidaného zboží. Stránka s obsahem košíku také slouží jako startovní bod v procesu vytváření objednávky. Odtud se mohu přesunout hned k zadávání osobních a dodacích údajů až k přehledu celé objednávky. Aplikace si po každém kroku ověřuje, jestli uživatel zadal všechna potřebná data, jinak nepůjde objednávku dokončit. Po přehledu objednávky započne proces odesílání objednávky, kdy jsou informace o objednávce odesílány na server a při jejím zdárném odeslání je jak nakupující, tak provozovatel informován prostřednictvím emailu, kde je jim zaslán přehled objednávky.




*Obrázek 7.4: Ukázka procesu vytváření objednávky*

Na základě tohoto přehledu objednávky je už pak na provozovateli, aby tuto objednávku vložil do svého primárního systému a dále zákazníka informoval o stavu objednávky. Tento systém tedy slouží jako alternativní nástroj na vytváření objednávek pro konkrétní e-shop. Nemůže ovšem fungovat jako samostatný nástroj pro evidenci objednávek do primárního systému provozovatele e-shopu.

## 7.4 Ověření funkčnosti

Aplikaci jsem testoval na celkem třech zařízeních s různými verzemi systému Android. Primárně byla aplikace testována na Sony Xperia Miro s verzí Androidu 4.0.3. Další zařízení byly Samsung Galaxy S3 s verzí Androidu 4.3 a Sony Xperia Z C6602 s verzí Androidu 4.4.4.

S testováním mi pomohli další 3 uživatelé, kteří dostali zařízení s aplikací a zkusili si s ní pracovat a uzavírat objednávky. Všichni 3 uživatelé už v minulosti někdy využívali webový nebo mobilní e-shop, takže orientace v aplikaci a její ovládání nebyl problém.

# 8 Závěr

Cílem této práce bylo vytvořit nástroj pro transformaci existujícího e-shopu do mobilních zařízení, aniž by bylo potřeba vytvářet novou aplikaci, ať už nativní nebo webovou, pro konkrétní e-shop. Já jsem se rozhodl pro univerzální nativní aplikaci komunikující se serverem. Jako zdroj dat posloužil XML feed, konkrétně ve formátu pro srovnávač cen Herureka.cz popsaný v kapitole 3. V kapitole 4 jsme si určili požadavky na systém a navrhli databázi. Dále jsme si popsali, jakým způsobem bude webová aplikace na serveru zpracovávat a ukládat data z XML feed do své interní databáze.

Kapitola 5 pojednávala o platformě Xamarin, na které výsledná mobilní aplikace běží. Popsali jsme si také UI nástrojovou abstrakci Xamarin.Forms pro multiplatformní psaní mobilních aplikací. V kapitole 6 jsme si popsali architekturu MVVM, na které naše aplikace funguje, a její jednotlivé komponenty. Popsali jsme si také rozdělení celé aplikace do jednotlivých vrstev a techniku synchronizace se serverem.

V kapitole 7 jsme si ukázali fungování celého systému od zápisu informací o e-shopu do webové aplikace až po uzavření objednávky na mobilní aplikaci.

Podařilo se mi tedy vytvořit komplexní systém, který umí na základě XML feed a dalších informací zadaných provozovatelem e-shopu naimportovat do nativní mobilní aplikace seznam produktů a umožnit tak uživateli objednat toto zboží jako na klasickém e-shopu. Je přitom potřeba jen minimální zásah do zdrojového kódu aplikace, kde je jen potřeba zadat adresu API a název aplikace.

Dalším možným vylepšením této práce je určitě rozšíření na další platformy, jako je iOS nebo Windows Phone, což je v tuto dobu z velké části splněno hlavně díky tomu, že většina kódu a všechny UI prvky jsou vytvořeny v PCL[7] projektu aplikace s využitím již zmíněného Xamarin.Forms. Dalším vylepšením by mohlo být vytvoření rozšířeného portálu jako webové aplikace. Daly by se také rozšířit možnosti, jak editovat vzhled aplikace (barva písma, barevné téma, umístění loga atd.). Jako celek by byl tento projekt využíván jako služba pro provozovatele e-shopů. Bylo by to levnější a jednodušší vytvoření nativní aplikace pro jejich obchod. Fungovalo by to na principu registrace uživatele provozujícího e-shop a žádosti o vytvoření aplikace s jím zadanými údaji. Proces tvorby této aplikace by tedy vyžadoval jenom ono minimum zásahů do zdrojového kódu aplikace.

[7] Portable Class Libraries – V tomto případě projekt se sdíleným kódem, který mohou využívat další projekty na různé platformy. [34]

# Literatura


[1] **RADWANICK, Sarah**. From Brick-and-Mortar to Mobile Click-and-Order: Which Retailers are Carving Out Space in the M-Commerce Market?. In: *ComScore* [online]. 2012 [cit. 2015-03-23]. Dostupné z: http://www.comscore.com/Insights/Press-Releases/2012/9/Retailers-Carving-Out-Space-in-the-M-Commerce-Market

[2] Web Application Development. *VTech Solution* [online]. 2006 [cit. 2015-03-23]. Dostupné z: https://vtechsolution.com/web-application-development/

[3] **BERNARD, Borek**. Úvod do architektury MVC. *Zdroják* [online]. 2009 [cit. 2015-03-12]. Dostupné z: http://www.zdrojak.cz/clanky/uvod-do-architektury-mvc/

[4] Model-View-Controller. *MSDN: the microsoft developer network* [online]. 2015 [cit. 2015-03-12]. Dostupné z: https://msdn.microsoft.com/en-us/library/ff649643.aspx

[5] **CASTLEDINE, Earle, Myles EFTOS a Max WHEELER**. *Vytváříme mobilní web a aplikace pro chytré telefony a tablety*. Brno: Computer Press, 2013. ISBN 9788025137635.

[6] **ŠTRÁFELDA, Jan**. Co je to E-shop. ŠTRÁFELDA, Jan. *Adaptic* [online]. 2005 [cit. 2015-02-28]. Dostupné z: http://www.adaptic.cz/znalosti/slovnicek/e-shop/

[7] Malý historický exkurz za prvními e-shopy. *ShopSys* [online]. 2010 [cit. 2015-02-28]. Dostupné z: http://www.shopsys.cz/clanky/maly-historicky-exkurz-za-prvnimi-eshopy/

[8] Specifikace XML pro internetové obchody. *Seznam nápověda* [online]. 1996 [cit. 2015-02-28]. Dostupné z: http://napoveda.seznam.cz/cz/specifikace-xml.html

[9] **HRUŠKA**, **Tomáš, Jan KROULÍK a Jiří TECHET**. *Internetové aplikace (WAP): III. část XML, XML schémata, XPath, XSLT*. Brno, 2012.

[10] Srovnávače zboží a cen. *Artikul* [online]. 2013 [cit. 2015-02-28]. Dostupné z: http://www.artikul.cz/obsah/Srovnavace-zbozi-a-cen

[11] Služby obchodům: Specifikace XML souboru. *Heureka.cz* [online]. 2007 [cit. 2015-03-01]. Dostupné z: http://sluzby.heureka.cz/napoveda/xml-feed/

[12] Vývoj metodikou RUP. In: *AspectWorks* [online]. 2015 [cit. 2015-03-23]. Dostupné z: http://www.aspectworks.com/development/jak-pracujeme/vyvoj-metodikou-rup

[13] **ZENDULKA, Jaroslav**. *Jazyk UML (Unified Modeling Language)*. 2003.

[14] **KŘENA, Bohuslav a Radek KOČÍ**. *Úvod do softwarového inženýrství: Studijní opora*. 2010.

[15] **ZENDULKA, Jaroslav a Ivana RUDOLFOVÁ**. *Databázové systémy IDS: Studijní opora*. 2006.

[16] *SQLite* [online]. 2000 [cit. 2015-03-11]. Dostupné z: https://www.sqlite.org/

[17] Get started with ASP.NET. In: *The ASP .NET Site* [online]. 2015 [cit. 2015-04-16]. Dostupné z: http://www.asp.net/get-started

[18] **MALÝ, Martin**. REST: Architektura pro webové API. In: *Zdroják* [online]. 2009 [cit. 2015-04-16]. Dostupné z: http://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/

[19] **PETZOLD, Charles**. *Creating Mobile Apps with Xamarin.Forms: Preview Edition*. Redmont, Washington: Microsoft Press, 2014. ISBN 9780735697256.

[20] **BANDARUPALLI, Kalyan**. Cross platform mobile development with Xamarin and Visual Studio. In: *Techbubbles: Microsoft Technlogy BLOG* [online]. 2015 [cit. 2015-03-23]. Dostupné z: http://www.techbubbles.com/microsoft/cross-platform-mobile-development-with-xamarin-and-visual-studio/

[21] **REYNOLDS, Mark**. Xamarin Mobile Application Development for Android. Birmingham: Packt Publishing, 2014. ISBN 9781783559169.

[22] Vyšší programovací jazyk. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2014 [cit. 2015-03-23]. Dostupné z: http://cs.wikipedia.org/wiki/Vy%C5%A1%C5%A1%C3%AD_programovac%C3%AD_jazyk

[23] **LANDWERTH, Immo**. .NET Core is Open Source. In: *MSDN Blogs* [online]. 2014 [cit. 2015-03-23]. Dostupné z: http://blogs.msdn.com/b/dotnet/archive/2014/11/12/net-core-is-open-source.aspx

[24] Xamarin.Forms Pages. In: *Developer Center - Xamarin* [online]. 2014 [cit. 2015-03-23]. Dostupné z: http://developer.xamarin.com/guides/cross-platform/xamarin-forms/controls/pages/

[25] Xamarin.Forms Layouts. In: *Developer Center - Xamarin* [online]. 2014 [cit. 2015-03-23]. Dostupné z: http://developer.xamarin.com/guides/cross-platform/xamarin-forms/controls/layouts/

[26] Shared Projects. In: *Developer Center - Xamarin* [online]. 2014 [cit. 2015-03-23]. Dostupné z: http://developer.xamarin.com/guides/cross-platform/application_fundamentals/shared_projects/

[27] **VICE, Ryan a Muhammad Shujaat** SIDDIQI. *MVVM survival guide for enterprise architectures in Silverlight and WPF: eliminate unnecessary code by taking advantage of the MVVM pattern - less code, fewer bugs*. Mumbai: Packt Publishing, 2012, vi, 465 p. ISBN 978-1-84968-342-5.

[28] 5: Implementing the MVVM Pattern Using the Prism Library 5.0 for WPF. In: *MSDN: The Microsoft developer network* [online]. 2014 [cit. 2015-03-27]. Dostupné z: https://msdn.microsoft.com/en-us/library/gg405484%28v=pandp.40%29.aspx

[29] **ALBAHARI, Joseph, Ben ALBAHARI a Peter DRAYTON**. *C# 5.0 in a nutshell*. 5th ed. Sebastopol: O'Reilly, c2012, xvi, 1042 p. In a nutshell (O'Reilly. ISBN 978-144-9320-102.

[30] Generic repository pattern using EF with Dependency injection (Ninject). In: *Code Project* [online]. 2013 [cit. 2015-03-23]. Dostupné z: http://www.codeproject.com/Tips/572761/Generic-repository-pattern-using-EF-with-Dependenc

[31] **FREEMAN, Adam a Steven SANDERSON**. *Pro ASP .NET MVC 3 Framework: Third Edition*. New York: Apress, 2011. ISBN 9781430234043.

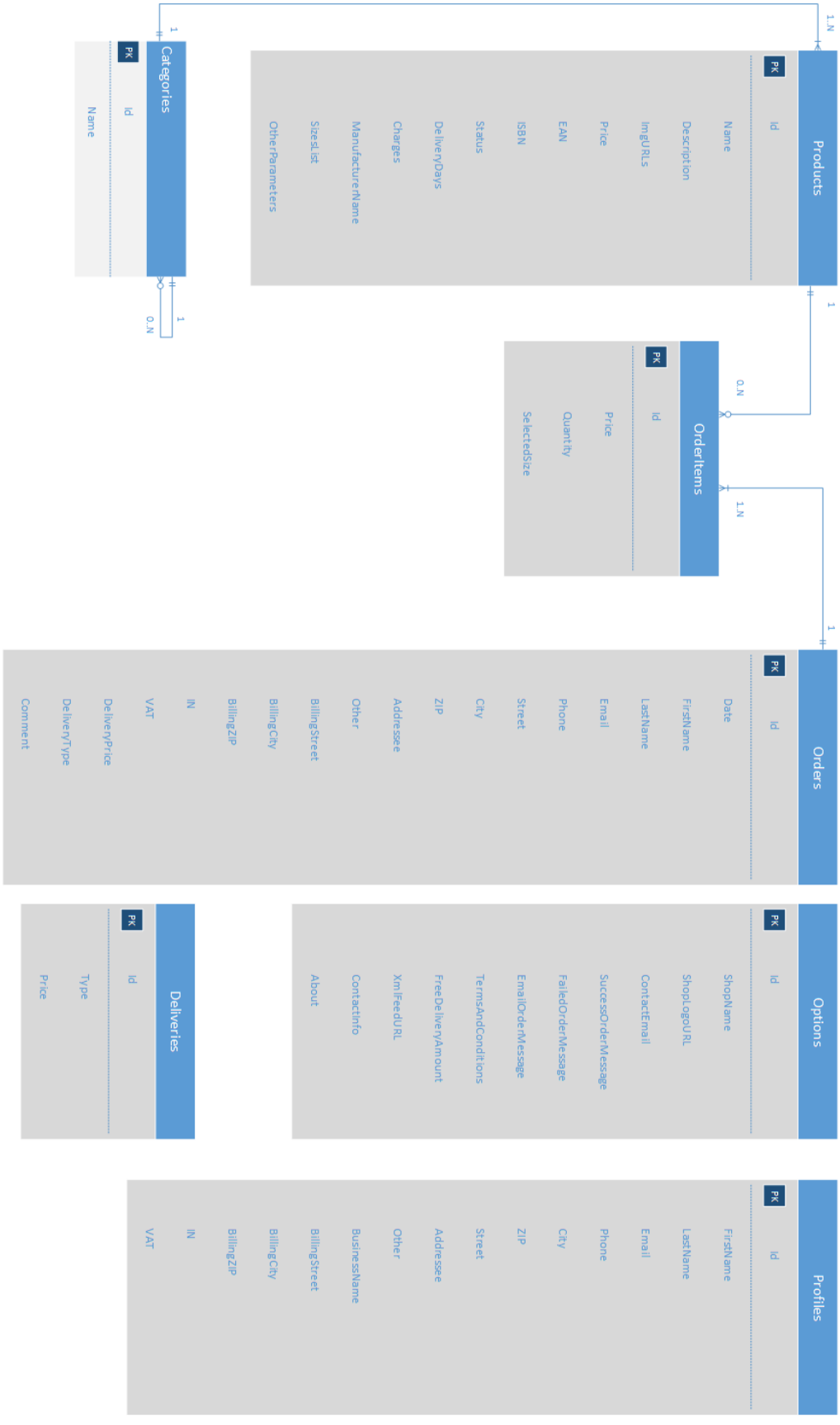
[32] *Ninject: Open source dependency injector for .NET* [online]. 2012 [cit. 2015-03-17]. Dostupné z: http://www.ninject.org/

[33] **KERR, Dave**. Commands in MVVM. In: *Code Project* [online]. 2012 [cit. 2015-03-25]. Dostupné z: http://www.codeproject.com/Articles/274982/Commands-in-MVVM

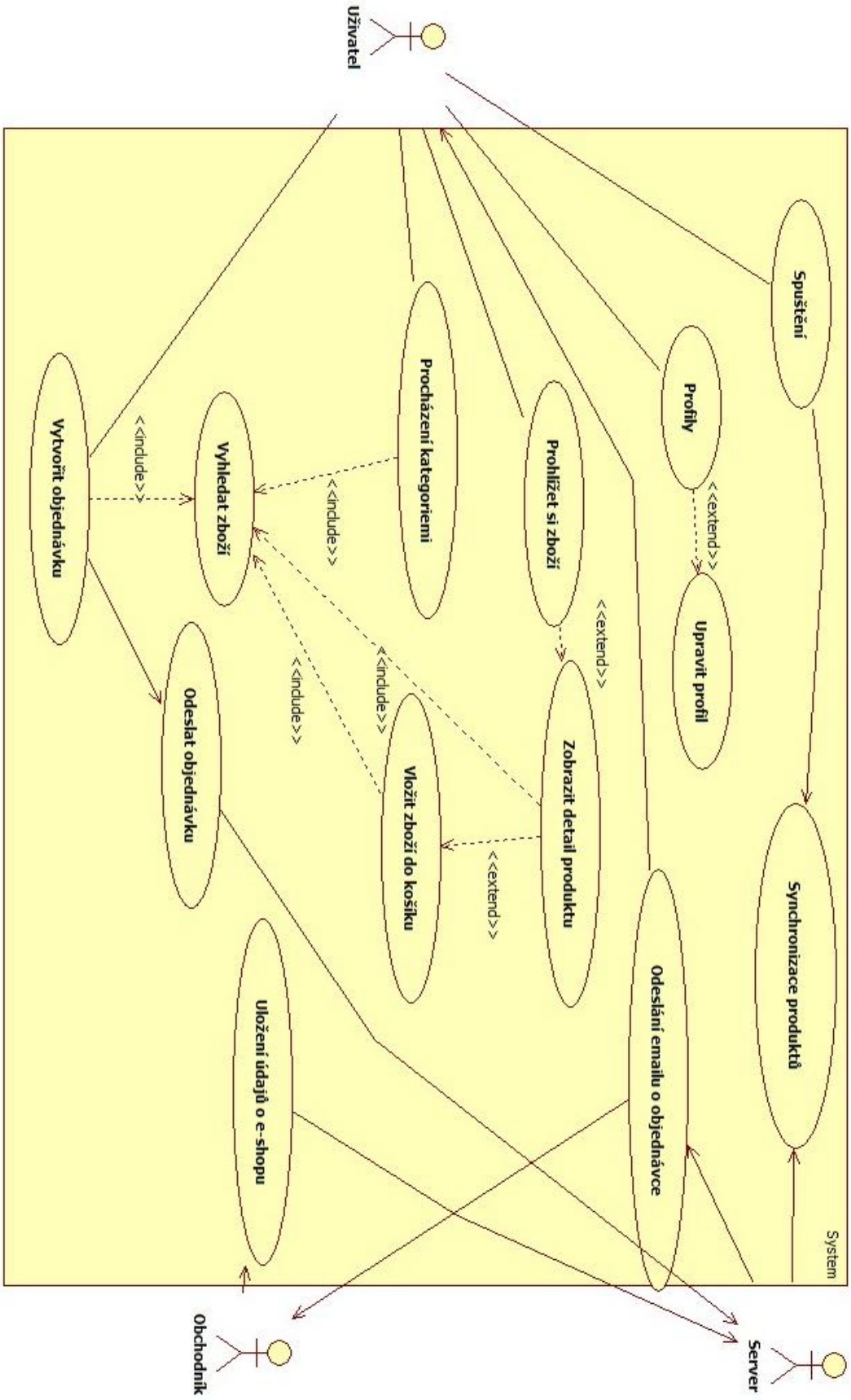
[34] Portable Class Libraries. In: *Developer Center - Xamarin* [online]. 2015 [cit. 2015-04-16]. Dostupné z: http://developer.xamarin.com/guides/cross-platform/application_fundamentals/pcl/

# Seznam příloh

# Příloha 1: ER Diagram databáze



# Příloha 2: Diagram případů užití



# Příloha 3: Detaily případů užití

| Případ užití: Synchronizace produktů |
| --- |
| ID: UC1 |
| **Účastníci:**<br>• Uživatel |
| **Vstupní podmínky:**<br>1. Aplikace má přístup k internetu.<br>2. Server je dostupný. |
| **Tok událostí:**<br>1. Systém zobrazí uživateli hlášku o probíhající synchronizaci databáze produktů.<br>2. Systém zkontroluje aktuálnost produktů v lokální databázi oproti databázi na serveru.<br>3. Systém zaktualizuje položky. |
| **Alternativní tok:**<br>1. Systém zobrazí hlášku o nedostupnosti serveru nebo neschopnosti připojení k internetu. |

| Případ užití: Prohlížet si zboží |
| --- |
| ID: UC3 |
| **Účastníci:**<br>• Uživatel |
| **Tok událostí:**<br>1. Uživateli se zobrazí katalog zboží. |
| **Alternativní tok 1:**<br>1. Aplikace nemá stažená žádná data o zboží.<br>2. Uživateli se zobrazí hláška, že je katalog prázdný. |

| Případ užití: Zobrazit si detail produktu |
| --- |
| ID: UC4 |
| **Účastníci:**<br>• Uživatel |
| **Vstupní podmínky:**<br>1. Uživatel má zobrazený neprázdný seznam produktů. |
| **Tok událostí:**<br>1. Uživatel klikne na daný produkt.<br>2. Zobrazí se detail daného produktu. |

| Případ užití: Vložit zboží do košíku |
|---|
| ID: UC5 |
| **Účastníci:**<br>• Uživatel |
| **Vstupní podmínky:**<br>1. Uživatel má zobrazený detail produktu, který chce vložit do košíku. |
| **Tok událostí:**<br>1. Uživatel klikne na tlačítko „Přidat do košíku“ v detailu produktu.<br>2. Systém uloží produkt do košíku a zobrazí hlášku o tom, že byl produkt přidán do košíku.<br>3. Uživatel pokračuje ve své činnosti. |
| **Výstupní podmínky:**<br>1. Podařilo se přidat produkt do košíku. |

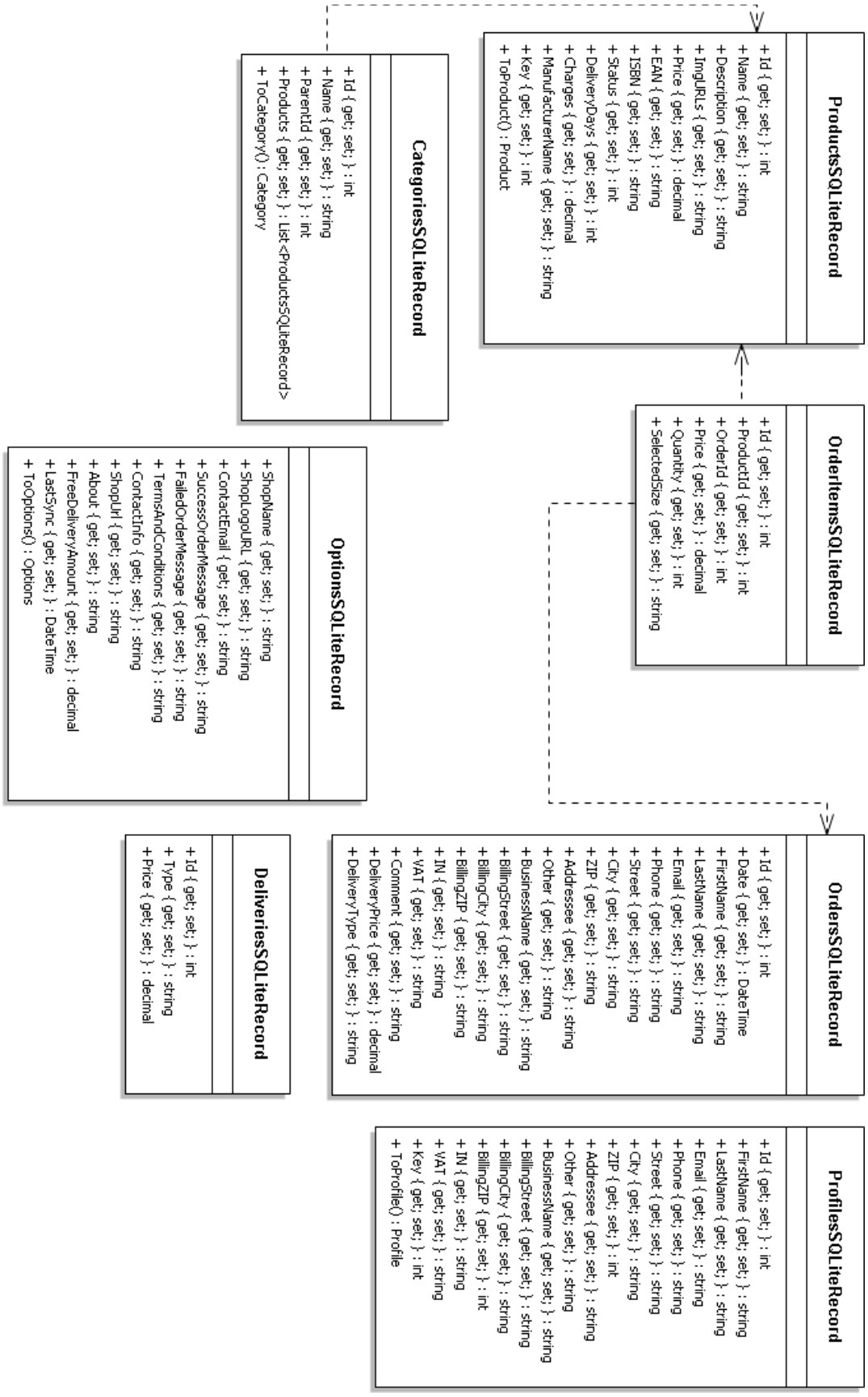
| Případ užití: Procházení kategoriemi |
|---|
| ID: UC6 |
| **Účastníci:**<br>• Uživatel |
| **Vstupní podmínky:**<br>1. Katalog produktů je neprázdný. |
| **Tok událostí:**<br>1. Systém nabídne uživateli strom kategorií<br>2. Uživatel si může vybrat danou kategorii/podkategorii a kdykoliv se přesunout na seznam produktů z dané kategorie/podkategorie |
| **Alternativní tok 1:**<br>1. Uživatel rozbalí podkategorie dané kategorie<br>2. Systém zkontroluje změny na serverové databázi vůči dané kategorii/podkategorii.<br>3. |
| **Alternativní tok 2:**<br>1. Uživatel rozbalí podkategorie dané kategorie<br>2. Systém zkontroluje změny na serverové databázi vůči dané kategorii/podkategorii. |

| Případ užití: Zobrazit košík |
|---|
| ID: UC7 |
| **Účastníci:**<br>• Uživatel |
| **Tok událostí:**<br>1. Uživatel klikne na ikonku košíku a zobrazí se mu obsah košíku. |

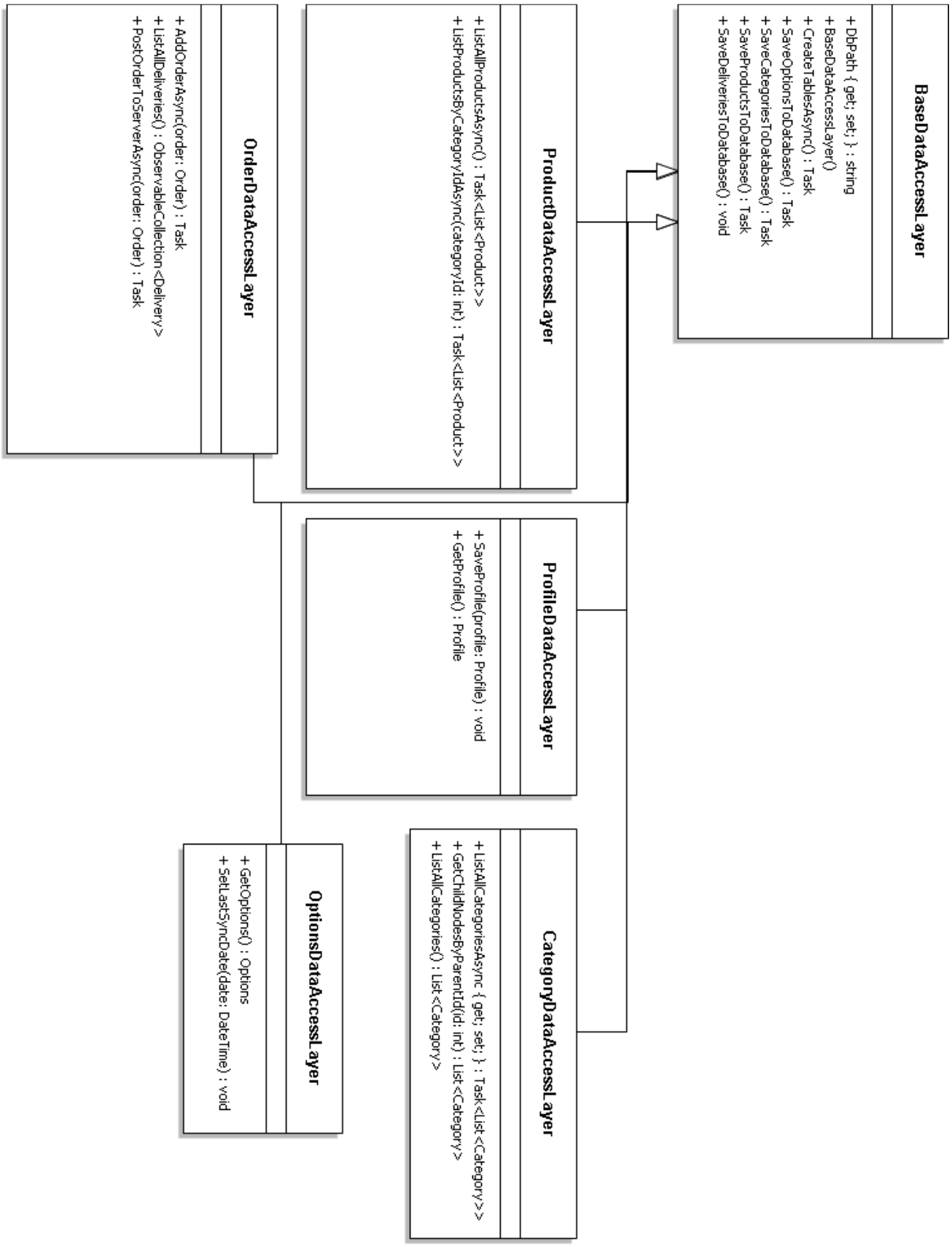
| Případ užití: Vytvořit objednávku |
| --- |
| ID: UC8 |
| **Účastníci:**<br>• Uživatel<br>• Server<br>• Obchodník |
| **Vstupní podmínky:**<br>1. Uživatel je na stránce s obsahem košíku.<br>2. Aplikace má přístup k internetu.<br>3. Server je dostupný. |
| **Tok událostí:**<br>1. Uživatel ze stránky z obsahem košíku postoupí dále k vytvoření objednávky.<br>2. Aplikace automaticky vyplní dodací údaje uložené v databázi.<br>3. Uživatel po vyplnění všech dodacích údajů a způsobu dopravy postoupí na poslední krok.<br>4. Aplikace zobrazí uživateli souhrn jeho objednávky a vyzve k závaznému uzavření.<br>5. Uživatel potvrdí objednávku.<br>6. Aplikace odešle objednávku a její položky na server, který si ho uloží do databáze.<br>7. Aplikace zobrazí hlášku o úspěšném dokončení objednávky a odešle mail s objednávkou obchodníkovi i uživateli.<br>8. Pokud se dodací údaje liší naproti těch v databázi, aplikace se zeptá, jestli chce uživatel nově zadané uložit.<br>9. Aplikace vyprázdní košík. |
| **Alternativní tok 1:**<br>1. Uživatel nevybere způsob dopravy v kroku 3.<br>2. Aplikace na to uživatele upozorní a nepustí ho k dalšímu kroku. |
| **Alternativní tok 2:**<br>1. Uživatel nevyplní povinné dodací údaje v kroku 3.<br>2. Aplikace na to uživatele upozorní a nepustí ho k dalšímu kroku. |
| **Alternativní tok 3:**<br>1. Aplikaci se nepodaří odeslat objednávku na server.<br>**2.** Aplikaci zobrazí hlášku o neúspěšném dokončení objednávky. |

| Případ užití: Profil |
| --- |
| ID: UC9 |
| **Účastníci:**<br>• Uživatel |
| **Tok událostí:**<br>1. Uživateli se zobrazí pole pro vyplnění dodacích údajů.<br>2. Uživatel po vyplnění tato data uloží do databáze. |

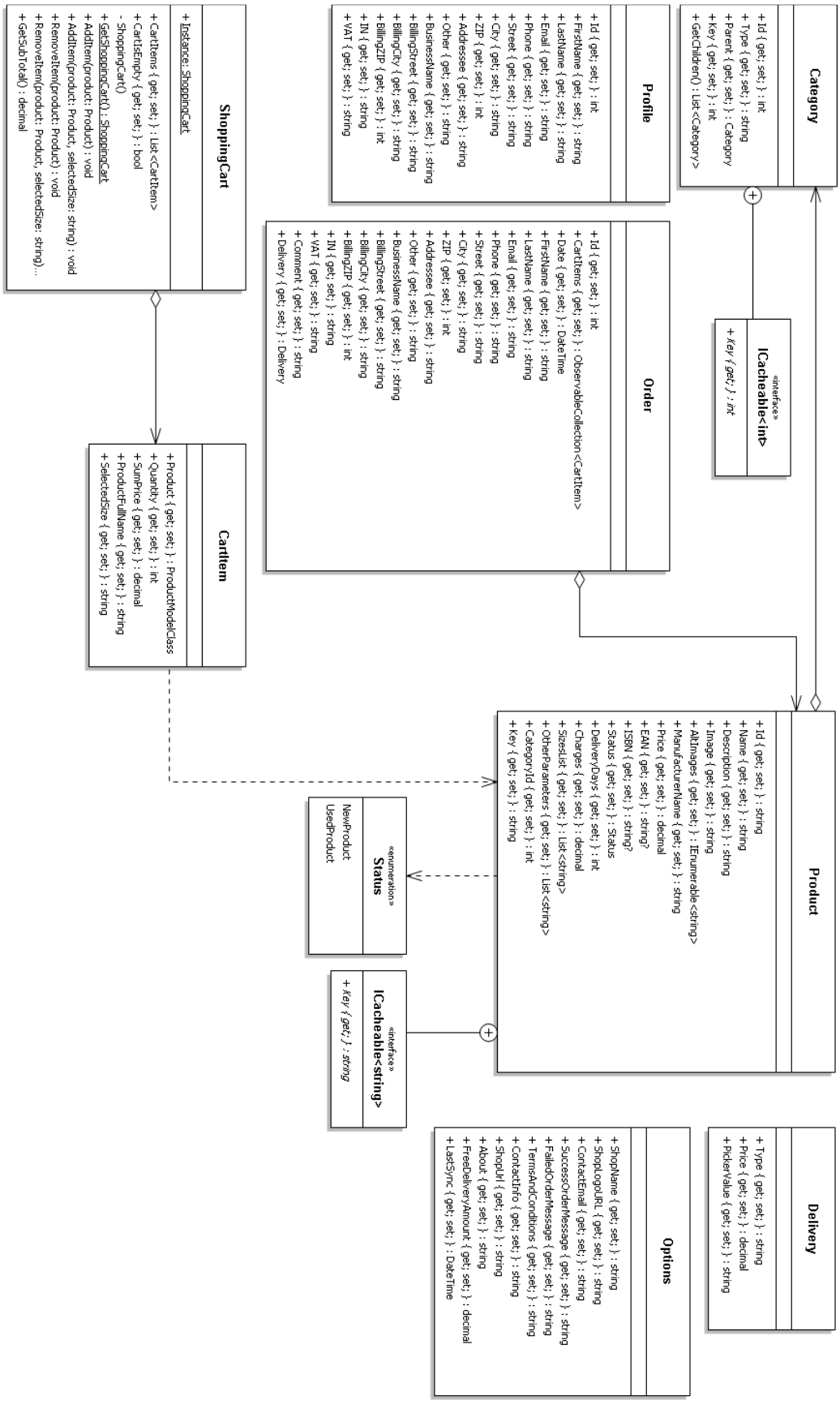
# Příloha 4: Diagram tříd SQLite Records



# Příloha 5: Diagram tříd Data Access Layer

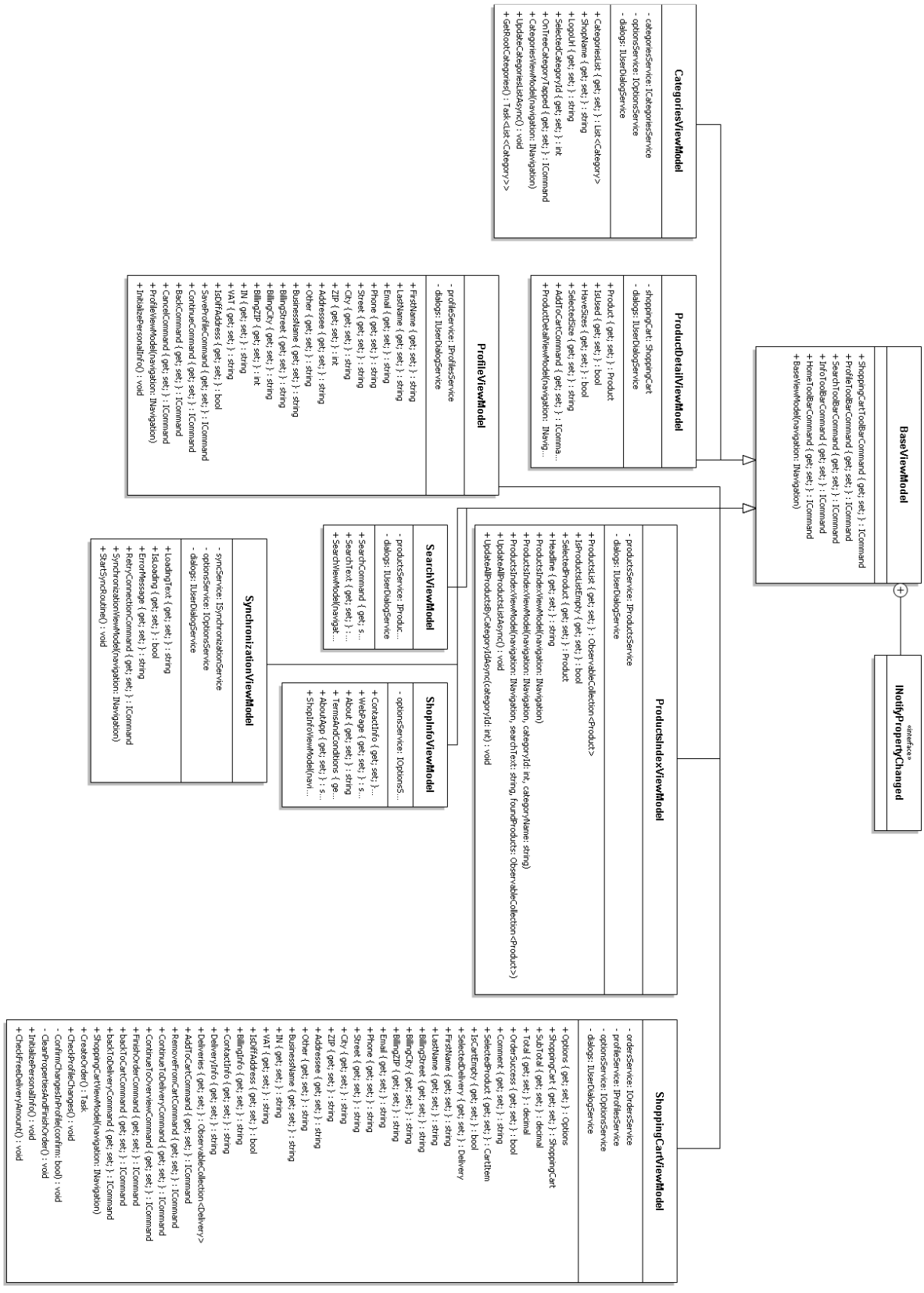


# Příloha 6: Diagram tříd Model Classes



# Příloha 7: Diagram tříd Data Service Layer

# Příloha 8: Diagram tříd ViewModels



# Příloha 9: Obsah CD

- /client/ – Složky se zdrojovými soubory a s dokumentací k mobilní aplikaci (klient)
- /diagrams/ - Výše přiložené diagramy ve formátech .pdf a .jpg
- /doc/ – Tato práce ve formátu .docx a .pdf
- /server/ – Složky se zdrojovými soubory a s dokumentací k webové aplikaci (server)