

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačů

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Petr Diviš**

Studijní program: Elektrotechnika a informatika (magisterský), strukturovaný
Obor: Výpočetní technika

Název tématu: **Modulární E-shop**

Pokyny pro vypracování:

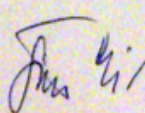
Seznamte se s problematikou elektronického obchodování. Navrhněte aplikaci e-shopu s využitím modulární architektury.
Prostudujte programovací jazyky používané pro webové aplikace a porovnejte jejich vhodnost pro implementaci e-shopu.
Navrženou aplikaci implementujte. Implementovaný systém důkladně otestujte a výsledky vyhodnoťte.

Seznam odborné literatury:

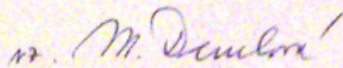
Dodá vedoucí práce

Vedoucí: Ing. Božena Mannová, Ph.D.

Platnost zadání: do konce letního semestru 2011/2012


doc. Ing. Miroslav Šnorek, CSc.
vedoucí katedry




prof. Ing. Boris Šimák, CSc.
děkan

V Praze dne 15. 2. 2011

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Diplomová práce
Modulární E-shop

Bc. Petr Diviš

Vedoucí práce: Ing. Božena Mannová, Ph.D.

Studijní program: Elektrotechnika a informatika, strukturovaný, Navazující
magisterský

Obor: Výpočetní technika

26. prosince 2011

Poděkování

V první řadě bych chtěl poděkovat své vedoucí Ing. Boženě Mannové, Ph.D. za vstřícný přístup, poskytnutí konzultací a její připomínky. Nesmím také zapomenout na Ing. Tomáše Černého, který vedl přínosná cvičení Architektury softwarových systémů a doporučil nám výbornou literaturu. Dále bych chtěl poděkovat své rodině za podporu a pochopení při studiu i při psaní této práce. V poslední řadě bych chtěl poděkovat všem, kteří pomáhali s opravou pravopisných chyb v této práci.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Rakovníku dne 1. 1. 2012

.....

Abstract

This thesis deals with a working concept of internet e-commerce application, which adapts modular software architecture. First the research on existing open-source and commercial solutions is done, then the research on programming languages and their development platforms with matching frameworks is performed. After choosing a suitable programming language with a framework the analysis and design of the implementation is described in depth. Then there are described the way of the implementation and necessary compromises when programming into the chosen language platform. After that the comparison with the existing solutions is done. In the conclusion results of the thesis are evaluated and compared with the defined goals.

Abstrakt

V této práci se zabývám internetovým obchodem, který si klade za cíl mít modulární softwarovou architekturu. Nejprve provádím rešerši mezi již existujícími komerčními i open-source internetovými obchody, dále provádím rešerši vhodných programovacích jazyků s vhodnými frameworky. Po zvolení vhodného programovacího jazyka s frameworkem provádím analýzu a návrh implementace. Poté popisuji způsob implementace a nutné kompromisy při programování do zvoleného jazyka. V dalším bodě práce provádím srovnání mého řešení s již existujícími řešeními. V předposledním bodě navrhuji další pokračování ve vývoji modulárního e-shopu. Na závěr hodnotím výsledky práce a porovnávám je s vytyčenými cíli.

Obsah

1	Úvod	1
1.1	Popis struktury práce	2
2	Popis řešeného problému	3
2.1	Popis problému	3
2.2	Vymezení cílů práce	4
3	Rešerše	5
3.1	Vymezení pojmů	5
3.2	Rešeršní zpracování vhodných implementačních prostředí	6
3.2.1	Jazyk PHP a jeho frameworky	6
3.2.2	Platforma Microsoft .NET	7
3.2.3	Platforma Java EE a frameworky	7
3.2.4	Perl, CGI	8
3.2.5	Python	8
3.2.6	Ruby on Rails	8
3.3	Rešeršní zpracování existujících implementací	8
3.3.1	Open-source řešení	9
3.3.1.1	NopCommerce	9
3.3.1.2	osCommerce	9
3.3.1.3	Quick cart	9
3.3.1.4	Zen.cart	9
3.3.1.5	Open cart	10
3.3.2	Komerční řešení	10
3.3.2.1	CubeCart	10
3.3.3	ShopCentrik	10
3.3.4	Shopio	10
3.3.5	oXyShop	10
3.3.6	Zoner inShop4	11
4	Analýza a návrh řešení	13
4.1	Požadavky uživatelů	13
4.1.1	Požadavky obchodníků	13
4.1.2	Požadavky zákazníka	14
4.2	Požadavky na hardware	14

4.2.1	Požadavky na hosting / vlastní server	15
4.3	Základní funkce	16
4.4	Požadavky z pohledu rozšiřitelnosti	17
4.5	Use Cases a Scénáře	18
4.5.1	Scénář 1 - Prohlížení zboží v obchodu	18
4.5.2	Scénář 2 - Registrace zákazníka	19
4.5.3	Scénář 3 - Nákup zboží	21
4.5.4	Scénář 4 - Přihlášení obchodníka	21
4.5.5	Scénář 5 - Vložení kategorie	22
4.5.6	Scénář 6 - Vložení zboží	23
4.6	Datový model	23
4.7	Návrhové vzory	24
4.7.1	Factory method	24
4.7.2	Plugin	24
4.7.3	Unit of Work	26
4.7.4	Repository	27
4.7.5	Inversion of Control	27
4.7.5.1	Dependency Injection	28
4.7.5.2	Service Locator	29
4.8	Architektura	29
4.8.1	Model View Controller (MVC)	30
4.8.2	Modularita v Model View Controller	31
4.9	Implementační prostředí	31
4.9.1	Rozšiřující knihovny pro platformu .NET	32
4.9.1.1	ASP.NET MVC 3	32
4.9.1.2	Entity Framework - Code First	32
4.9.1.3	SQL Compact Edition	33
4.9.1.4	Castle Windsor	33
4.9.2	Paged List	34
4.9.2.1	Glimpse	34
4.9.3	Ostatní použité technologie	34
5	Realizace	37
5.1	Komponenty aplikace	37
5.2	Použité části ASP.NET MVC 3	37
5.2.1	Global.asax	40
5.2.2	Controller	40
5.2.3	View	41
5.2.4	Model	42
5.2.5	Routování	43
5.2.6	Areas	44
5.2.7	Filtry	44
5.2.8	Validace dat	44
5.2.9	Atributy	44
5.2.10	Autorizace	45
5.2.11	Autentizace	45

5.3	Rozšiřující úpravy jádra	45
5.3.1	Propojení modelů s databází	45
5.3.2	Lokalizace	46
5.3.3	Frontend	48
5.3.4	Backend – administrace	48
5.3.5	Uživatelé	48
5.3.6	Atributy	49
5.4	Moduly	49
5.4.1	Start aplikace	49
5.4.2	Životní cyklus aplikace	49
5.4.3	Struktura modulu	50
5.4.4	Připojení modulů	51
5.4.5	Body rozšíření	52
5.4.5.1	Routy	52
5.4.5.2	Pohledy	52
5.4.5.3	Formuláře	53
5.4.5.4	Controllery	53
5.4.5.5	Služby	54
5.4.5.6	Menu	55
5.4.5.7	Widgety	56
5.4.5.8	Entity	56
5.4.5.9	Frontend	58
5.4.5.10	Backend – administrace	58
5.4.6	Omezení modulů	58
5.5	Spuštění aplikace	59
5.5.1	Instalace a odebrání modulu	59
6	Testování	61
7	Závěr	63
7.1	Zhodnocení	63
7.2	Rozšíření	63
A	Seznam použitých zkratk	67
B	UML diagramy	69
C	Screenshoty aplikace	71
D	Instalační a uživatelská příručka	73
D.1	Instalace prostředí	73
D.2	Instalace aplikace	73
D.3	Spuštění	73
D.4	Moduly	73
E	Obsah přiloženého DVD	75

Seznam obrázků

4.1	Use cases zákaznické části	19
4.2	Use cases administrační části	20
4.3	Základní datový model	25
4.4	Diagram tříd návrhového vzoru Factory method	26
4.5	Sekvenční diagram vytvoření pluginu	26
4.6	Sekvenční diagram registrace změněného objektu v Unit of Work	27
4.7	Závislosti ve vzoru Dependency Injection	28
4.8	Závislosti ve vzoru Service Locator	29
4.9	Model View Controller	30
5.1	Diagram komponent	38
5.2	Diagram komponent – část nižší úrovně	39
5.3	Základní struktura projektu	39
5.4	Životní cyklus požadavku	50
5.5	Struktura projektu modulu	51
5.6	Sekvenční diagram zobrazování widgetů	57
E.1	Struktura přiloženého DVD	75

Seznam tabulek

5.1	Možnosti umístění Widgetů	56
-----	-------------------------------------	----

Kapitola 1

Úvod

Internetových obchodů existuje nepřeborné množství. Nakupování na internetu se stalo fenoménem posledních 10ti let a vypadá to, že se bude jen nadále rozšiřovat. Jen v České republice se tržby internetových obchodů za poslední období Vánoc roku 2010 prodalo zboží za přibližně 13 miliard korun a průměrný nárůst obrátu oproti období Vánoc z roku 2009 byl neuvěřitelných 40% [1]. Od počátku internetového nakupování se toto odvětví neustále rozšiřuje a vyvíjí. Roste s ním nejen kvalita služeb nabízených obchodníky, ale i uživatelská přívětivost internetových obchodů a jejich systémů. To vše díky rychle se vyvíjejícím technologiím a silně konkurenčnímu prostředí internetového podnikání.

Pro tým popř. jednotlivce vyvíjející internetový obchod, ať už jako komerční či open-source produkt, je proto zásadní výběr programového vybavení a platformy, na které bude tento produkt zakládat a rozvíjet. Je třeba počítat s požadavky obchodníků a rychlým vývojem trhu s těmito softwarovými produkty. Nově vznikající webové standardy, inovované vlastnosti programovacích a grafických prostředí rychle prostupují mezi vývojáře a jen ti, kteří dokáží pružně reagovat a tyto novinky nabídnout ve svých produktech, udrží aktuálnost a konkurenceschopnost v moři jiných internetových obchodů.

Proč modulární?

Návrh a implementace individuálního řešení internetového obchodu obchodníkům se specifickými požadavky je sice v několika parametrech odlišná, ale hodně parametrů a funkcí má podobných. Práce programátora se tak velice často opakuje i přesto, že pro společné vlastnosti a funkce používá kód již hotový, funkční a otestovaný. Roste tak nejen časová náročnost vývoje takového řešení na míru, ale i náklady na vývoj, nemluvě o demotivaci programátora, který musí nevyhnutelně opakovat se rutiny provádět znova a znova.

Proto přišla myšlenka dynamicky rozšiřitelného internetového obchodu, který bude schopný rychlejšímu přizpůsobení požadavkům obchodníků a bude možné rozšířit jeho jádro o další funkce a technologie bez většího zásahu v kódu samotného jádra.

Již v úvodu je třeba předem říct, že tento program není vytvářen za účelem nasazení v komerčním prostředí, ale klade si za cíl ověřit, že je tvorba takového modulárního internetového obchodu možná a soustředí se na samotné možnosti rozšiřitelnosti jako takové na vhodně zvolené vývojové platformě.

Vývoj finálního produktu pro obchodníky vyžaduje mnohem delší časový interval, více lidských zdrojů, více zkušeností s vývojem internetových aplikací a nadměru kvalitní znalost vývojového prostředí, do kterého je nutné celý projekt dobře navrhnout a dlouhodobě otestovat, což je podle nabytých zkušeností nejlepší v ostrém produkčním prostředí. Vzhledem k tomu, že během této práce probíhalo seznamování se se zvoleným vývojovým prostředím, objevuje se v práci všeobecně známý vývojový fenomén "nalezení lepšího způsobu, když je hotovo".

1.1 Popis struktury práce

Tato práce představuje popis problému a vymezuje cíle a požadavky v kapitole 2. Kapitola 3 provádí rešerši na aktuální nabídku dostupných typových internetových obchodů jak komerčních, tak open-source. Dále provádí rešerši všech vhodných implementačních prostředí. Analýza v kapitole 4 se soustředí na systematický rozbor modulárního e-shopu do podstatných detailů, potřebných k nutnému porozumění fungování aplikace e-shopu. Včetně různých alternativ a volby implementačního prostředí z již v předchozí kapitole představených. V kapitole 5 je následně popsána implementace se zaměřením na nestandardní řešení. Implementace využívá vlastností zvolené implementační platformy a softwarové architektury, kterou dále rozšiřuje na modulární. Vytvořená aplikace je v kapitole 6 otestována, je vyzkoušeno nasazení na webový server a jsou změřeny hodnoty porovnávající rychlosti běhu základní aplikace a rychlosti aplikace s připojenými moduly.

Kapitola 2

Popis řešeného problému

V této kapitole je probrán problém návrhu elektronického obchodu a s ním spojené problémy a specifika elektronického obchodování. Kapitola popisuje nejen, co je cílem práce, ale i, co cílem práce není.

2.1 Popis problému

V současné době stále více obchodníků rozšiřuje svoje pole působnosti o internetové obchodování. Důvodem je snížení nákladů, lepší konkurenceschopnost a rozšíření obchodního radiusu na republikové, nebo dokonce celosvětové měřítko. Začínající maloobchodní prodejci nemohou dopředu určit vývoj prodeje zboží a požadavky na internetový obchod s tím spojené. Podnikatelé většinou neví, jak funguje elektronické obchodování a neznají svoje požadavky na internetový obchod.

Začínající prodejci mají proto před sebou složitou volbu. Nabízí se jim mnoho bezplatných open-source řešení, které si ovšem musí nainstalovat a spustit sami. Pokud ale nemají žádné zkušenosti s instalací takových řešení, musejí vyhledat pomoc od někoho, kdo má s takovými systémy zkušenosti. Často se jedná o řešení, které lze přizpůsobit individuálním požadavkům obchodníka pomocí modulů, nebo je možné funkčnost rozšířit doprogramováním systému programátorem. Další možností je pronájem hotového obchodu, který spravuje softwarová společnost. Pronajímatel tak platí určitý měsíční poplatek, za který dostane přístup k univerzálnímu obchodnímu systému se specifickou funkčností. Pokud nechce obchodník platit měsíční poplatek, může si většinou obchod koupit a získá tak přístup ke zdrojovým kódům a obchod si může nechat nainstalovat na vlastní server.

Podnikatelé se speciálními požadavky na internetový obchodní systém musí volit řešení v podobě tvorby internetového obchodu na míru. Jejich rozhodnutí spočívá ve volbě firmy, která se zabývá programováním těchto řešení.

Naopak společnosti zabývající se velkoobchodním prodejem mají přesné požadavky na internetový obchod. Často mají svoje vlastní IT oddělení, které vytváří a spravuje firemní internetový obchod, který zajišťuje prodej dalším firmám a volitelně i koncovým zákazníkům. Pro takové společnosti není problém přizpůsobovat systém měnícím se požadavkům.

V každém případě se obchodní model společností vyvíjí a mění a s ním se mění i požadavky na internetový obchodní systém. V případě změny požadavků je třeba reagovat změnou funkčnosti systému, kterou je třeba doplnit nebo změnit. Aby byla zaručena rozšiřitelnost a zůstaly nízké náklady, je vhodné zvolit modulární architekturu takového systému. Aplikace může být poté distribuována prodejci s jimi požadovanými funkcemi dodanými pomocí rozšiřitelných modulů, ale přitom bude mít stejné jádro.

2.2 Vymezení cílů práce

Cílem práce je navrhnout a implementovat aplikaci internetového obchodu s využitím vhodné modulární architektury. Případně bude třeba modularitu v nemodulární architektuře definovat. Výběr programovacího jazyka bude záviset na jeho vhodnosti pro webovou aplikaci obchodu a schopnosti implementovat danou modulární architekturu. Také bude brán zřetel na podporu a vývoj jazyka do budoucna a jeho sledování trendů. Implementace bude demonstrovat využití modularity aplikace záměnou různých modulů. Upřesnění požadavků na aplikaci bude provedeno analýzou.

Práce se nebude nezabývat řešením přívětivosti uživatelského rozhraní a jeho grafického vzhledu, protože to není jejím stěžejním tématem a bezpochyby by to vyžadovalo významné rozšíření rozsahu práce.

Také nebude cílem vývoj komplexního a vše zahrnujícího e-shopu, ale jen jeho jádra s moduly demonstrujícími funkčnost.

Kapitola 3

Rešerše

3.1 Vymezení pojmů

- **Architektura** je pojem předefinovávaný velmi často. Martin Fowler ve své knize [5] ani nedává přesnou definici. Jen říká, že mají dva společné prvky. A to - rozložení systému na části na nejvyšší úrovni a rozhodnutí, která se těžko mění. Také říká, že systém má ve většině případů více než jednu architekturu a pohled na to, co je architektonicky důležité, se mění během životního cyklu systému.

Podle článku IEEE Recommended Practice for Architectural Description of Software-Intensive Systems [9] je architektura uspořádání systému, rozděleného do komponent, jejich vztahů mezi nimi, mezi prostředím a principy vedoucími návrh a vývoj. Doporučuji také článek Petera Eelesa, který pojem softwarové architektury rozebírá podrobně ¹.

- **Modularita** je definována jako dekompozice systému do jednotlivých komponent a podsystémů, které jsou rozdělitelné a kombinovatelné[2]. Odpovídá to úrovni spárování komponent a zároveň stupni prolnutí, který pravidla systémové architektury dovolují vytvořit mezi komponenty systému. V softwarovém inženýrství se modularita vykládá jako úroveň oddělení a spolupráce komponent, nazývaných moduly, které jsou mezi sebou zaměnitelné. Moduly konceptuálně představují oddělení zájmů (SoC - Separation of Concerns) a definují logické hranice mezi komponenty. Systém propojuje moduly pomocí rozhraní. Každý modul má definované svoje rozhraní, přes které poskytuje a vyžaduje služby. Implementace zajišťuje napojení na rozhraní a funkčnost modulu. Modul má přístup k rozhraní jiných modulů. Koncepce modularity výrazně zlepšuje přehlednost a udržitelnost systému[2][20].
- **Návrhový vzor** systematicky pojmenovává, motivuje a vysvětluje všeobecný návrh, který cílí na opakující se návrhový problém v objektově-orientovaných systémech. Popisuje problém, řešení, kdy toto řešení lze aplikovat a jeho následky. Také poskytuje implementační rady a příklady. Řešení je obecné uspořádání objektů a tříd, které řeší daný problém. Toto řešení je upraveno a implementováno tak, aby řešilo problém v určitém prostředí[7]. Často se jedná o řešení využívající objektově orientovaného

¹<<http://www.ibm.com/developerworks/rational/library/feb06/eeles/>>

návrhu a je představováno obecnými objekty, třídami, jejich spoluprací a vztahy mezi nimi. Nedají se proto dobře využít ve funkcionálním programování. [18]

- **Framework** je předpřipravený softwarový systém, určený k dokončení a spuštění. Framework určuje architekturu pro skupinu podsystémů a poskytuje základní stavební bloky pro jejich vytvoření. Také definuje své části, které musí být přizpůsobeny pro dosažení požadované funkčnosti. V objektově orientovaném prostředí se framework skládá z *abstraktních* a *konkrétních tříd*. Spuštění takového frameworku spočívá ve skládání a dědění těchto existujících tříd. [2]. Podle Wikipedie je abstrakcí, ve které je běžný kód poskytující obecnou funkčnost často selektivně přepisován nebo konkretizován uživatelským kódem, což poskytuje požadovanou funkcionalitu. Frameworky jsou speciálním případem softwarových knihoven, které zahrnují znovupoužitelnost a jejich kód je odstíněn dobře definovaným API. Od běžných softwarových knihoven se frameworky liší následujícími vlastnostmi. Běh programu řídí framework a ne, část aplikace, která framework volá. Framework má vlastní výchozí chování a nejedná se o volání prázdných příkazů. Framework může být rozšířen uživatelem, který pomocí vhodného kódu určuje specifickou funkčnost. Vlastní kód frameworku není účelem upravovat ale rozšiřovat. [24]

3.2 Rešeršní zpracování vhodných implementačních prostředí

Webové aplikace jsou uloženy na serverech, které je poskytují klientům. Servery přijímají požadavky a většinou odpovídají zasláním stránek HTML v textové podobě, nebo jiných binárních dat, jako jsou obrázky, soubory, atd. Na serveru běží aplikace, která se o vyřizování požadavků stará. Podle programového vybavení serveru je třeba vybrat programovací jazyk či platformu, na kterých jsou webové aplikace postavené.

3.2.1 Jazyk PHP a jeho frameworky

Jde o skriptovací jazyk původně navržený pro vývoj dynamických webových stránek. Kód je vkládán do HTML obsahu a je interpretován za běhu, nedochází tedy k překladu. Ke svému běhu vyžaduje interpret, který bývá součástí webového serveru a generuje webové stránky za běhu z php skriptů. PHP procesor je dostupný pro většinu webových serverů na všech možných operačních systémech. Syntaxí byl PHP inspirován jazyky C, CGI, Perl a Python. I když šlo zpočátku o procedurální jazyk, od verze 3 je podporován i objektově orientovaný vývoj. Licenčně jde o volný software. [17]

PHP díky své dostupnosti, praktičnosti a rychlosti umožnil vývoj mnoha frameworků, které nabízejí snadnější vytváření komponent a navrhování struktur pro rychlejší aplikační vývoj. Mnohé frameworky existují dlouho a nabízejí funkčnost podobnou aplikačním platformám od Oracle a Microsoftu. Mezi známé a zavedené frameworky patří například Zend Framework, CakePHP, Symfony a také český Nette. Nejčastější webový server, na kterém je php provozováno je Apache spolu s kombinací databáze MySQL. Na serverech s PHP fungují i významné světové portály jako je Facebook, Wikipedia. Použitím PHP vzniklo také plno open source správců obsahu jako je MediaWiki, Drupal, Joomla, Wordpress, či Moodle. Nevýhodou PHP je, že neodděluje serverovou část od klientské a v základu je vývoj webových aplikací komplikovaný. [28]

3.2.2 Platforma Microsoft .NET

Dalším z často používaných programovacích nástrojů je kompletní vývojové prostředí od firmy Microsoft. Ta začala konkurovat jazyku PHP svým vlastním skriptovacím jazykem ASP, který později přenesla na vývojovou platformu .NET a přejmenovala jej na ASP.NET. ASP.NET je přeložena do bytekódu platformy .NET, který slouží jako mezistupeň překladu z vyšších programovacích jazyků platformy (ASP.NET, C#, Visual Basic.NET, J#, a další) do strojového kódu[15]. Díky společnému bytekódu lze programovat projekty pro webové prostředí i v ostatních jazycích .NET. Výhodou je také předkompilování do dll knihoven, které zrychlí běh aplikace, protože se nemusí opakovaně parsovat skripty jako u PHP. Dalšími výhodami je nativní podpora šablon, takže se redukuje duplikování kódu. Dále ASP.NET poskytuje velké množství ovládacích prvků a knihoven, které významně urychlí vývoj. Jako produkt Microsoftu je nejčastěji nasazován na serveru IIS v prostředí Windows, ale existují i další servery, které umožňují nasazení řešení díky dalším interesovaným firmám.[13]

ASP.NET nabízí k programování webových aplikací dva frameworky - WebForms a MVC. WebForms pracují na principu desktopového grafického rozhraní a snaží se obejít bezstavovost protokolu HTTP. Aplikace se řídí událostmi a zachovává stav. Používá se kombinace HTML a JavaScriptu pomocí dvou základních metod ViewState a SessionState. ViewState využívá posílání stavu ve skrytých polích formulářů, což může při špatném použití způsobit přenos nadměrného množství dat. Druhá metoda SessionState udržuje stav na serveru pomocí session, což může při nesprávném použití zatěžovat server.

MVC framework je klasická Model-View-Controller architektura připravená pro programátory přímo Microsoftem. Výhodou architektury MVC je, že se dá snadněji testovat. MVC se v posledních několika letech díky zaměstnancům Microsoftu velmi rychle vyvíjí a každý rok vychází nová verze. Aktuálně je dostupná MVC verze 3. Díky napojení na Entity Framework, což je framework pro mapování objektů z relací získaných většinou z databáze, se vývoj na této platformě velice urychlil [12].

3.2.3 Platforma Java EE a frameworky

Java platforma Enterprise Edition je často používaná platforma pro serverové aplikace napsané v jazyce Java. Je to edice obohacená o balíčky důležité pro aplikační server, které rozšiřují funkčnost a umožňují tak programátorům soustředit se na řídicí logiku místo integrování funkcí běžných na serverových aplikacích. Aplikace proto zvládají transakce, zabezpečení, škálovatelnost, paralelní běh a správu komponent, které se do aplikace přidávají.

Pro aplikační servery podporující Javu je také dostupné množství komponent, jak přímo od Oracle (dříve Sun Microsystems), tak od dalších organizací. Mezi nimi jsou například serverové komponenty Enterprise Java Beans, které umožňují tvorbu modulárních podnikových aplikací a oddělují business logiku aplikace od prezentační. Dále mezi ně patří Java Server Faces, což je MVC framework využívající XML pro oddělení uživatelského rozhraní a business logiky. Mezi aplikace od dalších organizací patří populární Spring a JBoss Seam frameworky, které jsou ekvivalentem pro nativní komponenty implementované v Java EE. Podporují současné techniky programování jako je inverzní řízení (ioc), aspektově orientované programování, objektově relační mapování, architekturu MVC a také unit testy [16][28].

Jako vývojové prostředí se nejčastěji používají open-source programy jako Eclipse IDE nebo NetBeans IDE, které buď plně podporují Java EE, nebo jsou pro ně dostupné plug-iny usnadňující vývoj. Komerčním programem je např. velmi kvalitní IntelliJ IDEA od společnosti JetBrains[19].

3.2.4 Perl, CGI

Perl je interpretovaný programovací jazyk pro tvorbu CGI skriptů již od roku 1987, od té doby ušel dlouhou vývojovou cestu a v roce 2000 vyšla 6. verze. Podobně jako PHP nepotřebuje kompilovat a je dynamicky typovaný. Podporuje všechny druhy programování, včetně funkcionálního a objektově orientovaného[21]. Perl nemá bohužel mnoho frameworků, které by usnadňovaly vývoj bussiness aplikací. Jedním z nejznámějších je Catalyst s architekturou MVC.

3.2.5 Python

Python je další dynamický interpretovaný programovací jazyk, který spatřil světlo světa před dvaceti lety. Je implementován v jazyce C. Velmi často je distribuován přímo v Linuxových distribucích, proto je známější Linuxovým uživatelům. Je v něm implementován aplikační server Zope, na kterém také běží některé z jeho frameworků, které jsou volně dostupné pro webové aplikace. Nejznámější je asi framework Django, který podporuje moderní metody vývoje a pomáhá vytvářet rychle aplikace, které jsou výkonné, čisté a neopakují se v nich kód[22].

3.2.6 Ruby on Rails

Ruby on Rails je webový framework, který vznikl na projektu Basecamp díky dánskému programátorovi David Heinemeier Hanssonovi. Framework je založený na jazyce Ruby a jeho architektura je opět MVC. Vše od Ajaxu v šablonách po napojení databáze v modelech je založeno na jazyce Ruby. Základní princip frameworku je "Convention over Configuration", což znamená, že se nejprve je nutné podívat jak věci fungují v základu a poté je případně změnit v konfiguraci. Framework také obsahuje routování URL na komponenty v aplikaci, návrhový vzor Active Record pro řízení dat a spolupracuje s javascriptovým frameworkem Prototype kvůli funkčnosti Ajaxu[23].

3.3 Rešeršní zpracování existujících implementací

Internet doslova přetéká řešeními pro internetové obchody, je jich nespočet zdarma k použití a ještě větší množství placených. Zde uvádím jen malý výběr povedených aplikací jak zdarma tak placených. Většina z nich je naprogramována objektově v jazyku PHP bez využití frameworků a to jim ubírá na přehlednosti při přidávání funkčnosti pomocí modulů, které některé z níže jmenovaných podporují. Mezi komerčními řešeními se jako programovací jazyk objevuje nejen PHP, ale i rozsáhlé webové aplikační platformy jako je Microsoft ASP.NET a Java Enterprise Edition.

3.3.1 Open-source řešení

3.3.1.1 NopCommerce

Jde o internetový obchod napsaný pro platformu Microsoft ASP.NET s použitím poslední verze MVC frameworku². Jeho výhoda spočívá v možnosti přidávání zásuvných modulů, které je třeba doprogramovat. I tak ale nabízí velké množství funkcí. Mezi klíčovými jsou například podpora změny vzhledu, vícejazyčnost, více měn, podpora SSL, reklamace, možnost sestavení produktů, import/export údajů, slevy, daně, různé platby, dopravy (hlavně americké), seznam oblíbených produktů a další a další. Jde o velice schopný modulární systém, s dobrou škálovatelností. Nyní se nachází ve verzi 2.2 a vývoj je stále aktivní díky komunitě z celého světa.

3.3.1.2 osCommerce

Velmi populární řešení internetového obchodu, které je zdarma dostupné jako volný software³. Naprogramováno je objektově v PHP s podporou databáze MySQL. Poskytuje komplexní řešení pro nabídku, prodej i administraci zboží. Velká vývojářská komunita přispívá do osCommerce tvorbou rozšiřujících modulů, kterých je přes 6 tisíc. Moduly jsou dostupné jako placené ale i zdarma. Nevýhodou je, že moduly je třeba instalovat manuálně a přepisovat kód jádra aplikace. Dokumentace popisující vytváření modulů je minimální.

3.3.1.3 Quick cart

Polská společnost Open Solution nabízí tuto aplikaci jako freeware, nebo v rozšířenějších placených verzích⁴. Quick cart je naprogramován v jazyku PHP, nevyžaduje SQL databázi a je neustále vyvíjen. V jeho základní verzi umožňuje kromě základních funkcí úpravu vzhledu pomocí šablon, doplnění dalších světových jazyků, výběr měn a další. Nevýhodou je, že nepodporuje tvorbu modulů. Na webu výrobce je dostupných pouze několik modifikací přidávajících funkčnost.

3.3.1.4 Zen.cart

Jde asi o nejrozšířenější open source řešení elektronického obchodu na trhu⁵. Aplikace je napsána objektově v jazyku PHP. Vzešel původně jako větev z osCommerce. Má velikou podporu mezi komunitními vývojáři a je přeložen do mnoha světových jazyků včetně češtiny. Podporuje rozšíření pomocí modulů, které ale mohou přepisovat kód jádra pro zajištění rozšířené funkčnosti. Vzhled se dá kompletně měnit pomocí šablonového systému, který se může někomu zdát celkem komplikovaný. Na oficiálních stránkách se ale nachází dokumentace s návody jak postupovat při změně vzhledu, jazyka nebo tvorbě modulů. Toto řešení dovoluje nasadit kompletní eshop i s informačními stránkami o společnosti, obchodních podmínkách a jiných. Samozřejmostí je správa pomocí administračního rozhraní.

²<http://www.nopcommerce.com/>

³<http://www.oscommerce.com/>

⁴<http://opensolution.org/>

⁵<http://zen-cart.com/>

3.3.1.5 Open cart

Velmi povedené řešení open source internetového obchodu s nativní podporou pluginů, jazykových překladů, šablon vzhledů, více světových měn⁶. Aplikace je opět napsána v jazyku PHP se skladováním dat v databázi MySQL. K aplikaci je poskytována jak podpora zdarma na diskuzním fóru, tak placená od partnerů, kteří nabízejí nasazení a kompletní správu.

Další volně dostupná řešení jsou například rozšíření pro známé CMS (systémy pro správu obsahu) jako jsou Drupal⁷ a Joomla⁸.

3.3.2 Komerční řešení

3.3.2.1 CubeCart

CubeCart⁹ je komerční řešení aplikace internetového obchodu chlubící se kvalitním produktem a velkým počtem zákazníků. Aplikace odporuje tvorbu vlastních šablon a zásuvných modulů. Množství hotových modulů lze dokoupit u vývojové firmy i u komunitních vývojářů. Aplikace je opět naprogramována v jazyku PHP.

3.3.3 ShopCentrik

Společnost NetDirect nabízí řešení internetových obchodů na klíč¹⁰ využívající stejné jádro a jejich produkt je velmi rozšířený zejména v českém prostředí. Jeho možnosti jsou velmi rozsáhlé a většinou se jedná o individuální řešení. Obchod je naprogramovaný na platformě Microsoft.NET a jeho služby využívají firmy jako Adidas, Best či Autobenex.

3.3.4 Shopio

Internetový obchod Shopio¹¹ nabízí svoje řešení napsané v jazyku PHP v několika variantách rozdělených podle množství funkcí a využívá databáze MYSQL. Jeho možnosti jsou podobné jako u jiných hotových řešení. Předností je integrace platebních metod tuzemských finančních domů a způsob dopravy je také připraven pro české obchodníky. SEO a marketingové nástroje jsou samozřejmostí.

3.3.5 oXyShop

Tento internetový obchod¹² je implementován na platformě Java EE a nabízí několik variant připravených řešení až po komplexní modulární systémy, které stojí miliony korun. Předností tohoto produktu je možnost škálovat výkon podle zatížení díky rozložení zátěže serverů do clusteru. Tento e-shop se může pochlubit oceněním Křišťálová Lupa a používá jej například známý prodejce výpočetní techniky Czech Computer.

⁶<<http://opencart.com/>>

⁷<<http://drupal.org/>>

⁸<<http://www.joomla.org/>>

⁹<<http://www.cubecart.com/>>

¹⁰<<http://www.shopcentrik.cz/>>

¹¹<<http://www.shopio.cz/>>

¹²<<http://www.oxyshop.cz/>>

3.3.6 Zoner inShop4

Internetový obchod Zoner inShop4¹³ pochází od známého českého softwarového a nakladatelského domu Zoner. Je vyvíjený na platformě Microsoft .NET a má již 11ti letou historii. Také nabízí napojení všech možných účetních systémů, plateb a dopravců. Neustále se vyvíjí a je aktualizovaný. Jednou z jeho mnoha referencí je například Plzeňský Prazdroj s napojením na interní firemní systém SAP. Licence nabízí formou měsíčních poplatků.

¹³[<http://www.inshop.cz/>](http://www.inshop.cz/)

Kapitola 4

Analýza a návrh řešení

Tato kapitola analyzuje požadavky na aplikaci a možnosti zvolené platformy vzhledem k implementaci obchodu. V závislosti na analýze souběžně tato kapitola představuje návrh řešení aplikace. Požadavky jsou rozděleny tak, aby korespondovaly se zadáním. Tedy na základní požadavky, které jsou potřebné pro jádro obchodu a dále na pokročilé požadavky, které jsou vhodné pro rozšíření aplikace pomocí modulů.

4.1 Požadavky uživatelů

Uživatel e-shopu je každý obchodník, správce, zaměstnanec obchodníka, který se o e-shop stará, a hlavně každý zákazník, který obchod navštíví. Z pohledu jeho potřeb je důležité zajistit mu jednoduchý přístup a rychlou orientaci. Další podsekcce se zabývají oběma rolmi obchodníka a zákazníka zvlášť.

4.1.1 Požadavky obchodníků

Z hlediska přístupu potencionálního obchodníka k aplikaci internetového obchodu lze rozdělit jeho potřeby na základní a pokročilé. Pro realizaci základní aplikace obchodník potřebuje zpřístupnit prezentaci svého zboží, umožnit jeho nákup a doplnit informace o svém podnikání, kontakty, obchodní podmínky a další texty. Celý elektronický obchod musí také mít neveřejnou část, ke které má přístup jen obchodník a která je chráněná přístupovými údaji, přihlašovacím jménem a heslem. V této administrační neveřejné části může spravovat všechno zboží, provedené objednávky a informační texty.

Pro minimální běh aplikace potřebuje alespoň spravovat zboží a vyřizovat objednávky. Jako další rozšíření pro obchod splňující doporučení a zároveň certifikaci organizace APEK¹ můžeme přidat již zmiňované důležité obchodní informace.

Pokud by obchodník chtěl prodávat ve více zemích, nebo získat zákazníky mluvícími jinými jazyky, bude požadovat i vícejazyčnost. S tou se do seznamu požadavků dostávají další funkce umožňující práci s cenami v jiných měnách.

¹<http://www.apek.cz/8482/2061/clanek/certifikacni-pravidla/>

Dalšími specifickými požadavky mohou být například napojení na firemní účetnictví a různé daňové sazby a jiné manipulace s cenami. Což se týká i napojení na skladové systémy a propagaci skladových zásob na stránky s informacemi o zboží. Následné platby a způsoby dodání mohou vyžadovat integraci s platebními systémy bankovních ústavů a systémy spedičních společností, se kterými má obchodník uzavřené smlouvy.

V dnešní době patří mezi časté požadavky obchodníků také integrace obchodu do webových služeb porovnávajících ceny zboží, díky kterým se mohou propagovat a přitáhnout tak zákazníky do svého obchodu.

Předpokládaný pozitivní obchodní vývoj prodejce s sebou nese rozvoj požadavků obchodníka na elektronické obchodování, proto je na místě nabídnout další rozšíření jeho starší aplikace podle jeho specifických požadavků. Tyto požadavky je možné realizovat napojením dalších modulů do stávající aplikace bez toho, aby se musel zásadně měnit datový model obchodu. Takže chceme aplikovat rozšíření formou nadstavby místo zásadní přestavby a vývoje nové aplikace abychom zamezili děláním stejných věcí vícekrát a přicházeli tak o čas a peníze. V případě druhé strany, tedy obchodníka, určitě nestojíme o investování do stejné věci dvakrát.

Z důvodu bezpečnosti a integrity aplikace mu není vhodné povolit jako laikovi zasahovat do integrace rozšiřitelných funkcí - modulů. Jako kompromis mu můžeme povolit upravovat některá nastavení modulů.

Za specialitu lze považovat možnost volby, kde se budou jednotlivé funkční prvky nacházet a případná změna vzhledu. I když tato funkce patří do univerzálních systémů pro správu obsahu a jiných univerzálních řešení "na vše".

4.1.2 Požadavky zákazníka

Zákazník přichází na webové stránky proto, aby zjistil informace o produktu a případně produkt zakoupil. To je základní požadavek na aplikaci. Další specifikace těchto požadavků se týkají různých způsobů dodávky a platby za zboží. Další rozšíření je vylepšení uživatelského komfortu, zavedení možnosti komunikace s obchodníkem a diskuse s dalšími zákazníky, kteří se také o produkt zajímají, tedy sociální funkce. Můžeme také zavést zákaznické volby pro práci se zbožím jako jsou hledací a filtrační funkce, nebo ukládání zboží do vlastních seznamů. Ještě větším rozšířením může být zavedení nebo integrace webů sociálních a jiných prodejních služeb. Samozřejmostí je možnost prohlížení historie objednávek a případných reklamací.

V případě vícejazyčného webu zákazník určitě ocení možnost převodu mezi měnovými kurzy. Zrovna tak je v dnešní době zvykem informovat zákazníka o stavu objednávky pohybu zboží nejen formou emailových, ale i SMS zpráv.

4.2 Požadavky na hardware

Z hlediska hardwarových požadavků je třeba probrat problém z několika úhlů. Nejprve musíme rozhodnout jestli je nutné se vůbec touto otázkou zabývat, protože většina aplikací internetových obchodů běží na sdílených hosting serverech třetích firem bez větších problémů.

Proto lze požadavky přetransformovat na požadavky na hosting či vlastní server.

4.2.1 Požadavky na hosting / vlastní server

Rozhodnutí zda zvolit hosting nebo si pořídit vlastní server záleží na požadavcích zákazníka a na mohutnosti e-shopu. Zásadními parametry jsou také zvolená implementační platforma, požadavky podpory, prostorové a databázové parametry, garantovaná doba dostupnosti a technická podpora.

Hosting lze rozdělit do 3 kategorií.

- **Sdílený web hosting** je webový prostor poskytovaný hostingovou firmou na serveru, který hostuje desítky až tisíce jiných webových aplikací, které mezi sebou sdílí zátěž serveru. Toto se hodí pro nenáročné aplikace, které navštíví pár desítek uživatelů denně. Tento hosting má omezené parametry podle platformy, kterou hostuje a programu, který má zákazník zaplacený. Pro velké aplikace je tato varianta hostingu hodně omezující.
- **Virtuální server hosting** už je lepší varianta poskytovaná také hostingovou firmou. Dává zákazníkovi neomezené možnosti co se nastavení parametrů týče. Hardwarový server je sdílený většinou mezi dvěma až deseti jinými zákazníky a proto aplikace dostává k dispozici více systémových prostředků jako je výpočetní výkon a operační paměť potřebné pro bezproblémový běh náročnějších aplikací. Tyto prostředky jsou zpravidla pevně vyhrazené. Výhodou tohoto hostingového programu je lepší technická vybavenost serveru a možnost instalovat vlastní rozšiřující balíčky do systému včetně pravidelných aktualizací hostingovou firmou. Každý zákazník má totiž svůj virtuální operační systém. Virtuální servery mají také tu výhodu, že na nich zákazník může spustit více svých aplikací pod různými doménami s vlastními databázemi a dalšími službami.
- **Dedikovaný server hosting** je nejpokročilejší varianta, kterou mohou hostingové firmy nabídnout. Jedná se o vlastní hardwarový server, který je pouze pro jednoho zákazníka, který si na něm může spouštět a nasazovat vlastní řešení a je jen na něm, co se serverovým výkonem podnikne. Taková varianta je nejdražší z programů hostingů a proto je vhodná pro náročné zákazníky očekávající klidně stovky přístupů za vteřinu.

Výhodou všech těchto programů je servis serverů a řešení potíží hostingovou firmou, která garantuje dostupnost a poskytuje technickou podporu ať už v ceně nebo extra placenou.

Dalšími variantami jsou hosting na cloudu, který balancuje rozložení požadavků na více fyzických serverů v případě náhlých výkyvů zátěže. Jde v současné době o nastupující trend, kterého využívají korporátní zákazníci popřípadě si sami taková řešení fyzicky zřizují.

Poslední variantou je umístění vlastního fyzického serveru do housingového centra. Výhodou je úplná kontrola nad serverem a to i fyzická. Nevýhodou je nutná znalost potřebných parametrů při pořizování hardware a také při instalování a udržování systému. Pokud se jedná o jediný server udržovaný zákazníkem jsou potřeba zkušenosti a znalosti pro údržbu a zabezpečení serveru.

[25]

Dále jsou možné další méně časté varianty umístění aplikace na jiné aplikační servery. Příkladem budiž různé linuxové distribuce na diskových serverech NAS nebo na domácích

routerech. Tato řešení ale nejsou optimalizovaná na běh složitějších aplikací a proto je lze považovat za nouzová nebo pro osobní potřeby.

4.3 Základní funkce

Základními funkcemi jsou myšleny funkce, které musí být obsaženy už v jádře internetového obchodu. Bez těchto funkcí by obchod nemohl operovat a další rozšíření by nebylo možné, protože by nebylo na čem stavět.

E-shop by nemohl fungovat bez věcí již zmíněných v sekci 4.1. Některé nepotřebují rozšiřitelnost, ale některé ano a je třeba je pro ni připravit. Tím se bude zabývat další sekce 4.4.

Do základu e-shopu patří:

- **frontend** – část pro zákazníky, určená pro procházení a prodej zboží. Obsahuje vybrané informace viditelné veřejně všem návštěvníkům webu.
- **backend** – zaheslovaná část pouze pro pověřené osoby, tedy správce a obchodníky. Obsahuje všechna nastavení a správu dat, také prodejní informace atd.
- **autentizace správců** – důležité zabezpečení backendu přihlášením uživatele s přihlašovacím jménem a heslem. Nepovolané osoby se tak do této části e-shopu nedostanou.
- **lokalizace** – obchodu do více jazyků. V základu je bez nutnosti překladu využíván jediný hlavní jazyk aplikace.
- **autentizace zákazníků** – přihlašování zákazníků při nákupu, aby mohli zadat svoji adresu a další iniciály pro urychlení vyřízení objednávek pro příští nákup. Zároveň jsou chráněna i informace, které si předává obchodník se zákazníkem jako jsou informace o objednávkách a reklamách.
- **autorizace** – rozlišení přístupových práv zákazníků a správců po přihlášení do systému obchodu.
- **katalog zboží** – základní katalog zboží zajišťující přehledné třídění zboží do kategorií.
- **produktová karta** – základní informace o zboží, které budou zastupovat jednotlivé produkty (kód zboží, název, cena, popis ...).
- **informační stránky** – další stránky, které neposkytují přímo informace o zboží, ale jsou důležité pro informování zákazníka o nákupu, obchodních podmínkách, kontaktech na obchodníka, atd.
- **objednání** – standardní proces objednání zboží, díky kterému získá obchodník informace od zákazníka, které zboží objednává, jak ho chce doručit a jakým způsobem za něj zaplatí.
- **přihlašovací formulář** – základní přihlašovací formulář, který se zobrazí nepřihlášenému uživateli, který chce provést nějakou ze zabezpečených operací jako například objednání zboží.

- **menu** – standardní menu v hlavičce, které poskytuje základní orientaci při navigaci po webových stránkách obchodu. Může obsahovat například odkazy na obchodní informace, kontakty, obsah nákupního košíku, či katalog zboží.
- **registrace zákazníka** – důležitý registrační proces, který umožní uchování dat od zákazníka v e-shopu a poskytne je zákazníkovi opět při přihlášení pomocí zvoleného přihlašovacího jména a hesla.
- **nákupní košík** – povinná vlastnost e-shopu, chováající se jako skutečný nákupní košík, do kterého v obchodě vkládáme vybrané zboží, abychom ho poté mohli zaplatit u pokladny. V případě e-shopu ovšem potřebujeme vědět nejen jaké je v košíku zboží, ale i způsob dopravy k zákazníkovi a způsobu platby, viz. *objednání*.
- **přehledné URL** – v dnešní době nezanedbatelná věc, takzvané *routování*. URL obsahující lidmi a vyhledávači čitelné položky, např.
<http://www.obchod.cz/kategorie/ovoce/zbozi/banany>.

4.4 Požadavky z pohledu rozšiřitelnosti

V této části probereme možné body rozšíření. Pokud bychom chtěli dosáhnout co nejuniverzálnějšího řešení, byly by tyto body skutečně všude a aplikace by se zkomplikovala, ztratila by přehlednost a pravděpodobně by mnohonásobně i klesl výkon. Proto jsem vybral jen z mého pohledu nejpraktičtější body rozšíření, díky kterým zůstane aplikace přehledná a výkonná.

V aplikaci byly vzaty v úvahu tyto body rozšíření:

- **zboží** – Informace o zboží budou rozšiřitelné o jakékoli položky. Zároveň by byla vhodná možnost vytváření variant zboží jako jsou barvy a velikosti. Samozřejmostí je dostupnost informací o zboží v modulech.
- **kategorie** – stejně jako zboží, informace o kategoriích budou rozšiřitelné a dostupné v modulech. Například o cesty k ilustračním obrázkům.
- **menu** – do menu v hlavičce bude možnost vkládat další položky s odkazy na informace a jiné doplňkové funkce.
- **administrace** – v administraci půjde spravovat nastavení a informace obsažené v modulech, pokud budou obsahovat administrační část. Tato část v modulech bude také zabezpečena.
- **layout** – rozložení položek z modulů na stránce v různých částech obchodních procesů, tak jak si zvolí tvůrce modulu.
- **vícejazyčnost** – do e-shopu půjde nastavit libovolný počet překladů limitovaný jen přeložením výrazů do všech jazyků.
- **zákaznický účet** – data ze zákaznického účtu aktuálního zákazníka budou k dispozici každému modulu.

- **data** – všechna data budou dostupná pro moduly a zároveň půjdou data z modulů vkládat do databáze aplikace a rozšiřovat tak datový model.
- **vzhled** – grafický vzhled se bude dát mocet měnit nezávisle na aplikační logice. Jeho vrstva bude oddělená od databáze i od samotné aplikace.
- **šablony** – se vzhledem úzce souvisí šablony, které budou rozšiřitelné bez potřeby změny aplikační logiky.
- **routování** – tedy adresy URL, které odkazují na jednotlivá místa v aplikaci ať už stránky nebo funkčnosti. Tyto adresy budou moci být nastavitelné přímo na místa v modulech.

4.5 Use Cases a Scénáře

Základní use cases pro pevné jádro jsou popsány v následujícím use case diagramu v Obrázku 4.1, v Obrázku 4.2 a také následně ve scénářích.

4.5.1 Scénář 1 - Prohlížení zboží v obchodu

Role:

- Neregistrovaný zákazník
- Systém

Předpoklady:

Zákazník se nachází na vstupní stránce internetového obchodu.

Důsledky:

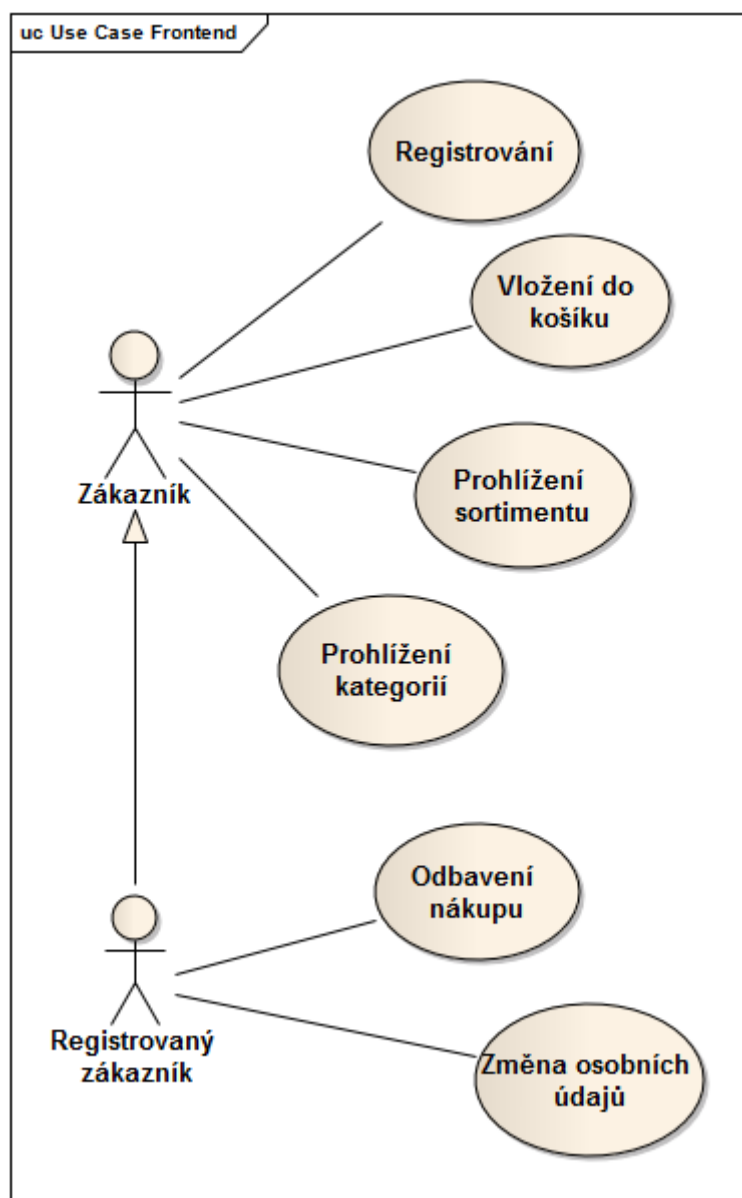
Zákazník získal informace, které hledal.

Hlavní příběh:

1. Zákazník klikne na kategorii zboží.
2. Zákazník klikne na odkaz s názvem zboží, které ho zajímá.
3. Zákazníkovi se zobrazí informace o zboží.

Možná rozšíření:

- 1.1 Zákazník vyhledá zboží pomocí hledání
- 2.1 Zákazník vyfiltruje zboží pomocí parametrů
- 3.1 Zákazník klikne na odkaz "Koupit" a tím vloží zboží do košíku.

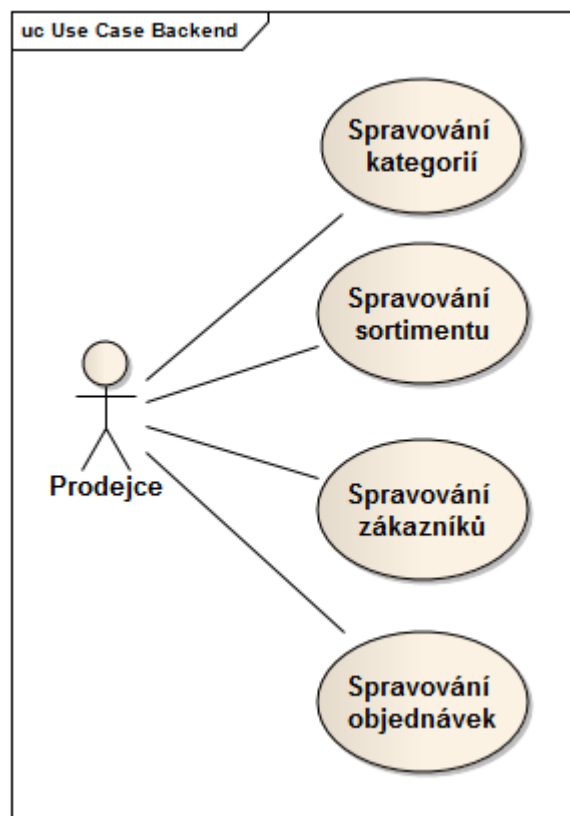


Obrázek 4.1: Use cases zákaznické části

4.5.2 Scénář 2 - Registrace zákazníka

Role:

- Neregistrovaný zákazník
- Systém



Obrázek 4.2: Use cases administrační části

Předpoklady:

Zákazník není přihlášený.

Důsledky:

Zákazník je registrovaný v systému.

Hlavní příběh:

1. Zákazník klikne na odkaz "Registrovat".
2. Zákazník vyplní osobní údaje.
3. Zákazník potvrdí registrační údaje.

Rozšíření:

- 3.1 Zákazník se odhlásí kliknutím na odkaz "Odhlásit".

4.5.3 Scénář 3 - Nákup zboží

Role:

- Neregistrovaný zákazník
- Systém
- Správce obchodu

Předpoklady:

Zákazník je přihlášený a nachází se na stránce produktu.

Důsledky:

Zákazník má potvrzeno, že je zboží objednané.

Hlavní příběh:

1. Zákazník klikne na odkaz "Koupit" a vloží tak zboží do košíku.
2. Zákazník zkontroluje obsah košíku a klikne na odkaz "Objednat".
3. Zákazník vyplní údaje a zvolí způsob dopravy a platby.
4. Zákazník zkontroluje údaje a odešle je kliknutím na tlačítko "Potvrdit".
5. Zákazník dostane potvrzení od systému, že objednávka byla přijata.

Rozšíření:

- 1.1 Zákazník vloží do košíku další zboží.
- 2.1 Zákazník si přečte obchodní podmínky pod odkazem "Obchodní podmínky".
- 5.1 Správce obchodu kontaktuje zákazníka v případě komplikací s objednávkou.

4.5.4 Scénář 4 - Přihlášení obchodníka

Role:

- Nepřihlášený správce
- Systém

Předpoklady:

Správce se chce přihlásit do administrace a zná administrační přihlašovací jméno a heslo. Nachází se na úvodní stránce aplikace.

Důsledky:

Správce je přihlášený a může spravovat administraci.

Hlavní příběh:

1. Správce klikne na odkaz "Přihlásit".
2. Správce zadá přihlašovací údaje do načteného formuláře a potvrdí jej.
3. Správce klikne na odkaz "Administrace".

Rozšíření:

- 3.1 Správce se odhlásí kliknutím na odkaz "Odhlásit".

4.5.5 Scénář 5 - Vložení kategorie**Role:**

- Přihlášený správce
- Systém

Předpoklady:

Správce je přihlášený a nachází se na hlavní stránce administrace.

Důsledky:

V katalogu je nová kategorie.

Hlavní příběh:

1. Správce klikne na odkaz "Katalog".
2. Správce klikne na odkaz "Vložit kategorii".
3. Správce vyplní informace o nové kategorii a zvolí nadřazenou kategorii.
4. Správce potvrdí informace kliknutím na tlačítko "Vložit".

Rozšíření:

- 4.1 Správce si zobrazí kategorii v katalogu.

4.5.6 Scénář 6 - Vložení zboží

Role:

- Přihlášený správce
- Systém

Předpoklady:

Správce je přihlášený a nachází se na hlavní stránce administrace.

Důsledky:

V katalogu je nové zboží.

Hlavní příběh:

1. Správce klikne na odkaz "Produkty".
2. Správce klikne na odkaz "Vložit produkt".
3. Správce vyplní informace o novém zboží a zvolí nadřazenou kategorii.
4. Správce potvrdí informace kliknutím na tlačítko "Vložit".

Rozšíření:

- 3.1 Klikem s podrženým CTRL může zvolit správce více kategorií.

4.6 Datový model

Z důvodu uchování informací o zboží, o zákaznících, o objednávkách a dalších; je třeba zavést základní datový model, který se bude moci dále rozšiřovat.

Na Obrázku 4.3 jsou znázorněné důležité entity propojené vazbami mezi sebou. Diagram pochází z aplikace Visual Studio, ve které byl e-shop kompletně naprogramován včetně návrhu datového modelu, viz. 4.9. Z diagramu vyplývá že, že datový model je pouze minimální. Základní entity jako je **BasicProduct** (zboží) a **BasicCategory** (kategorie) obsahují minimum vlastností pro reprezentaci v e-shopu. Kategorie je organizována ve stromové struktuře katalogu. Zboží je obsaženo v kategoriích a volitelně může být obsaženo ve více kategoriích.

Zboží se při nákupu ukládá uživateli do košíku **CartItem** s informacemi o množství a datumu. Při vytvoření objednávky se položky z košíku přesunou do detailu objednávky **OrderDetail** a ty se pak agregují do samotné entity objednávky **Order** spolu s informacemi o zákazníkovi. Volitelně, pokud je zákazník přihlášený se objednávka přiřadí k přihlášenému zákazníkovi **Customer**.

Poslední důležité entity jsou zdroje **Resource** obsahující překlady do více jazyků a nastavení **Setting**, které uchovává obecná nastavení aplikace e-shopu.

Samozřejmostí je možnost tento základní datový model rozšířit pomocí modulů. Toto je detailně probíráno v Kapitole 5.

4.7 Návrhové vzory

Pro rozsáhlou aplikaci navrženou v objektově orientovaném programovacím jazyce jsou návrhové vzory nutností. Neustále se opakující problémy při návrhu složitých aplikací jako je tato vyžadují analýzu všech známých návrhových vzorů. V následujících sekcích si představíme všechny vhodné vzory a probereme jejich klady a zápory a důvody proč byly či nebyly vybrány. Všechny vzory jsou čerpány z literatury [7] a [5].

4.7.1 Factory method

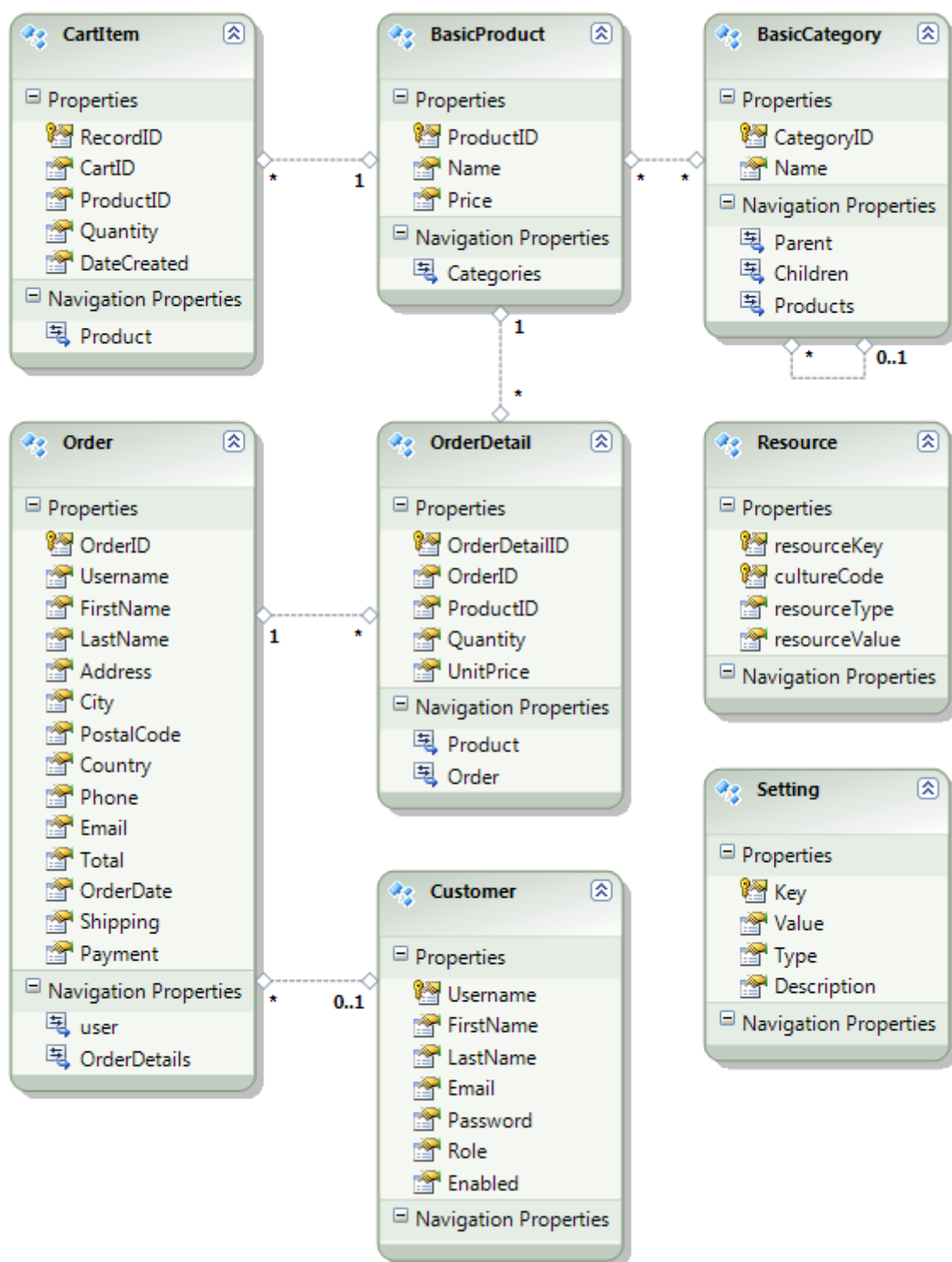
Tento návrhový vzor se uplatní při vytváření instancí různých implementací rozhraní. Patří mezi vytvářecí návrhové vzory. Jeho využití je probíráno už v dalším vzoru *Plugin*. Využití vzoru *Factory method* spočívá v tom, že volaná metoda je továrnou na instance, které nemusí být ve třídě specifikované, ale využívají se pouze jejich předpisy v podobě rozhraní nebo abstraktních tříd. To přináší do kódu flexibilitu. Samotná metoda také nemusí být specifikovaná, její konkrétní podobu jí mohou dát až zdědění potomci. Další volitelnou funkčností této metody může být inicializace až na žádost v době, kdy je instance třídy potřeba.

Diagram tříd 4.4 z [7] popisuje způsob tvorby konkrétních instancí ve *factory method*. Třída **Konkrétní tvůrce** již obsahuje tělo metody **factory method**, která vytváří třídu **Konkrétní Produkt**. Ten implementuje rozhraní **Produkt**, jehož předpis zná třída **Tvůrce** a v metodě **operace** s ním pracuje. Samozřejmě ve finále tuto metodu spouští až třída **Konkrétní Tvůrce**, pro zjednodušení v diagramu metoda uvedena podruhé není.

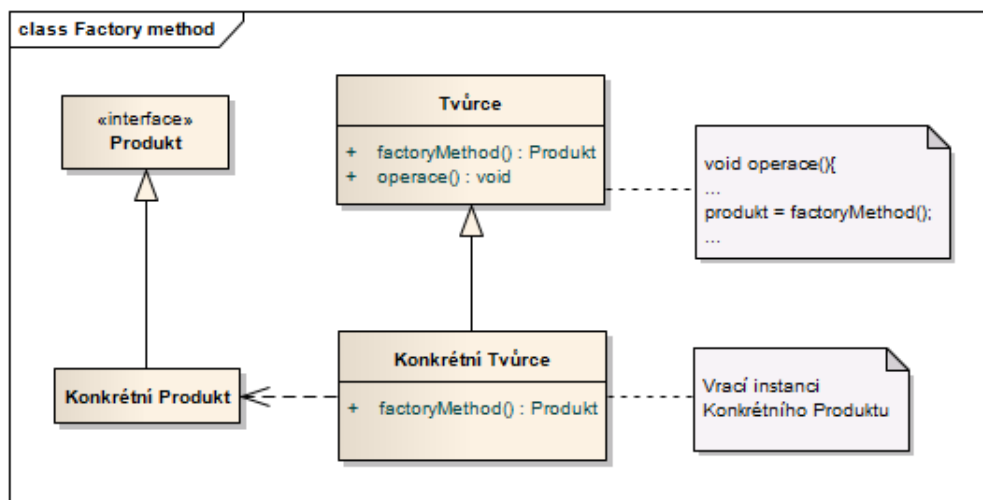
4.7.2 Plugin

David Rice a Matt Foemmel popisují návrhový vzor *pluginu* jako vhodný prostředek pro odstínění různých implementací rozhraní. Pluginy se dají měnit za jiné přímo za běhu aplikace, bez potřeby překompilovávat aplikaci. Výměnu pluginu za jiný indikuje *konfigurace*. Návrhový vzor plugin je v tomto případě velice jednoduchý a zakládá se na vytvoření instance z jedné implementace rozhraní. Volání implementovaných členů rozhraní se provádí po vytvoření instance implementace rozhraní například pomocí vzoru *Factory method*. Toto popisuje sekvenční diagram z [5] - viz Obrázek 4.5. Na tomto přístupu injektáže implementací rozhraní lze založit základy modulů v aplikaci e-shopu, viz dále.

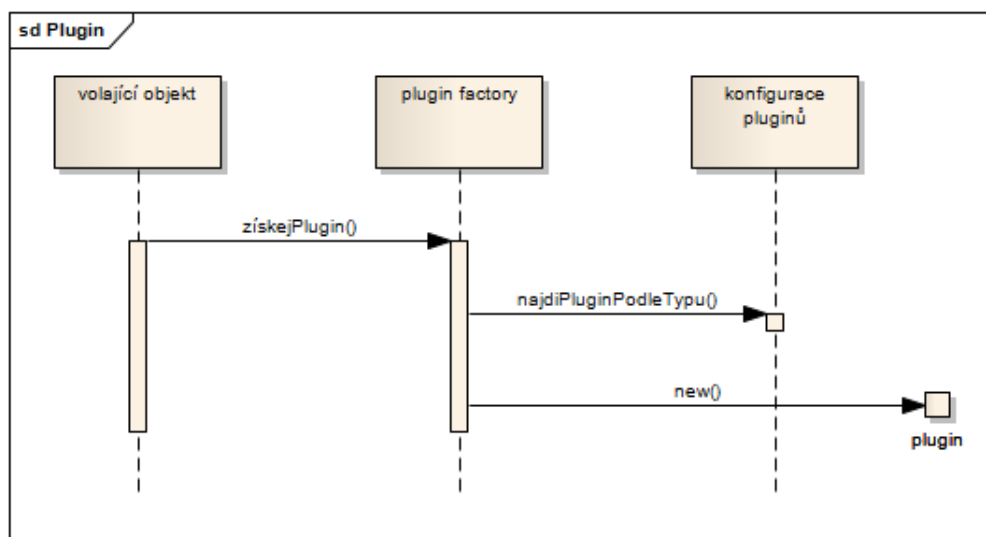
Způsob konfigurace se liší implementací od implementace, ale jednoduchou formou konfigurace je textový soubor XML s tagy označujícími konkrétní plugin, třeba formou cesty v souborovém systému ke zkompileované dynamické knihovně. Další formou konfigurace může být konvence, podle které se budou platné pluginy nacházet v určitém adresáři aplikace.



Obrázek 4.3: Základní datový model



Obrázek 4.4: Diagram tříd návrhového vzoru Factory method



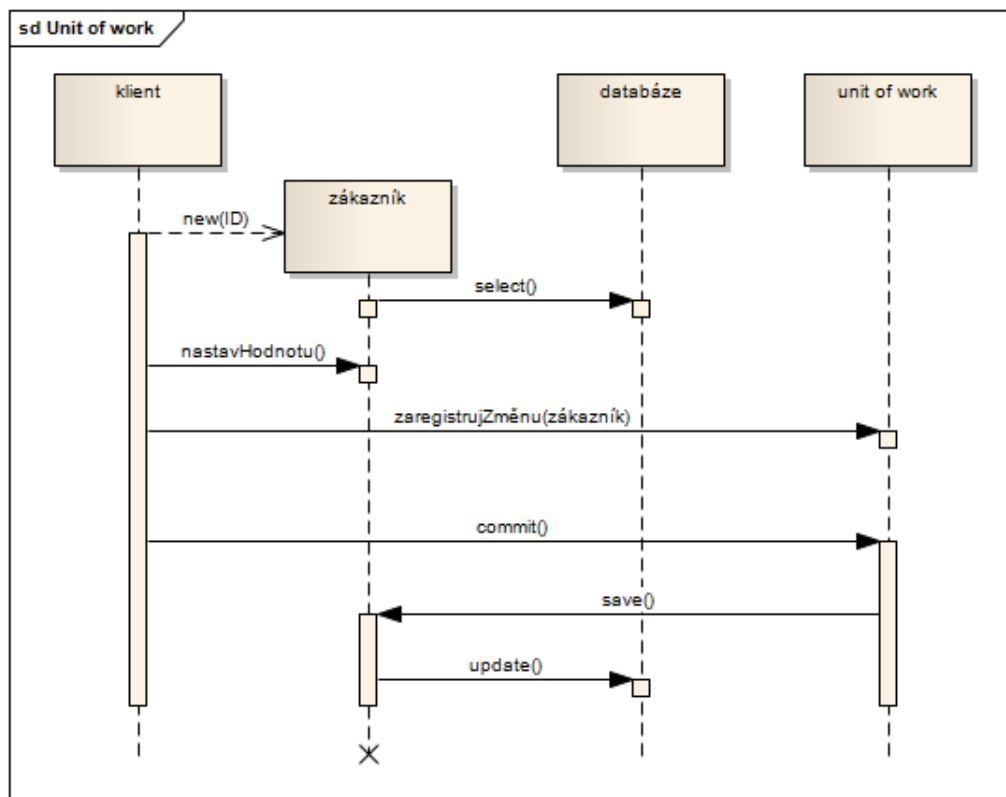
Obrázek 4.5: Sekvenční diagram vytvoření pluginu

4.7.3 Unit of Work

Tento návrhový vzor se hodí na udržování seznamu datových objektů, které byly změněny aplikací a vyřizuje jejich správné zapsání do databáze a zároveň řeší konflikty při paralelních přístupech. Vzorek se pro aplikaci hodí vzhledem k objemu předpokládaných transakcí s databází. Zvolená ORM vrstva ho bude obsahovat již implementovaný.

Diagram 4.6 popisuje registraci změněného objektu v Unit of Work. Tuto registraci musí uživatel při změně objektu provést sám. Toto je volitelné proto, aby objekt Unit of Work nemusel zbytečně zapisovat všechny změny, které se do databáze promítnout nemusejí.

Objekt Unit of Work takto registruje všechny nové, změněné nebo smazané objekty pro pozdější promítnutí do databáze.



Obrázek 4.6: Sekvenční diagram registrace změněného objektu v Unit of Work

4.7.4 Repository

Jedná se o další návrhový vzor pro odstínění přístupu k databázi. Jeho předností je, že umožní plně objektově orientovaný způsob dotazování a všechno spojené s dotazy na databázi zapouzdří v sobě. Různé datové zdroje tak mohou být odstíněny za vrstvou potřebného počtu *repozitářů*. Programátor používající *repository* se tak může plně soustředit na práci s objekty, které potřebuje. Další výhodou je, že je konkrétní *repository* vyměnitelné za jiné, které poskytuje jiná data nebo nepřistupuje k databázi. Toto je vhodné pro unit testy. Urychlí se tak běh testů. Zároveň je to perfektní příprava pro *Dependency Injection*, které představíme dále. Další informace lze nalézt v [5].

4.7.5 Inversion of Control

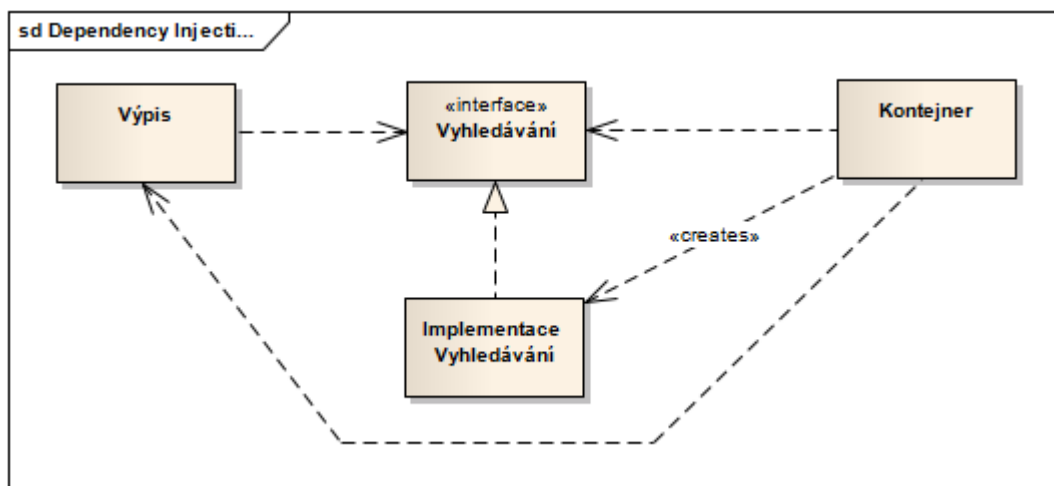
V současnosti velmi používaná metoda pro ovládání aplikace, která řeší řízení aplikace z jednoho místa. Článek Martina Fowlera celou problematiku výstižně popisuje [11]. Toto je stručný extrakt z článku, kde je celá problematika *Inversion of Control* probírána detailněji.

Zabývá se v něm také rozdíl mezi souvisejícími návrhovými vzory *Dependency Injection* a *Service Locator*.

Inversion of Control neboli česky *převrácení ovládní*, je často používaný mechanismus ve frameworkích, které řídí program svojí hlavní programovou smyčkou a programátor se jen stará o doplňování funkcí a specifikací abstraktních tříd. Dříve bylo zvykem mít v programu vlastní hlavní ovládací smyčku. Pod pojmem *Inversion of Control* se dá představit spousta věcí a spousta z nich bude správná, proto byly zavedeny již zmíněné úzce zaměřené vzory *Dependency Injection* a *Service Locator*, které si nyní ve stručnosti představíme.

4.7.5.1 Dependency Injection

Tento návrhový vzor spočívá ve vkládání závislostí - konkrétních rozhraní - do tříd pomocí řídicí třídy, která zná všechny potřebné informace. Řídící třída se většinou nazývá "kontejner". Obrázek 4.7 vzor názorně ilustruje. Separátní objekt *Kontejner* obsadí proměnnou ve třídě *Výpis* s vhodnou implementací rozhraní *Vyhledávání*.



Obrázek 4.7: Závislosti ve vzoru Dependency Injection

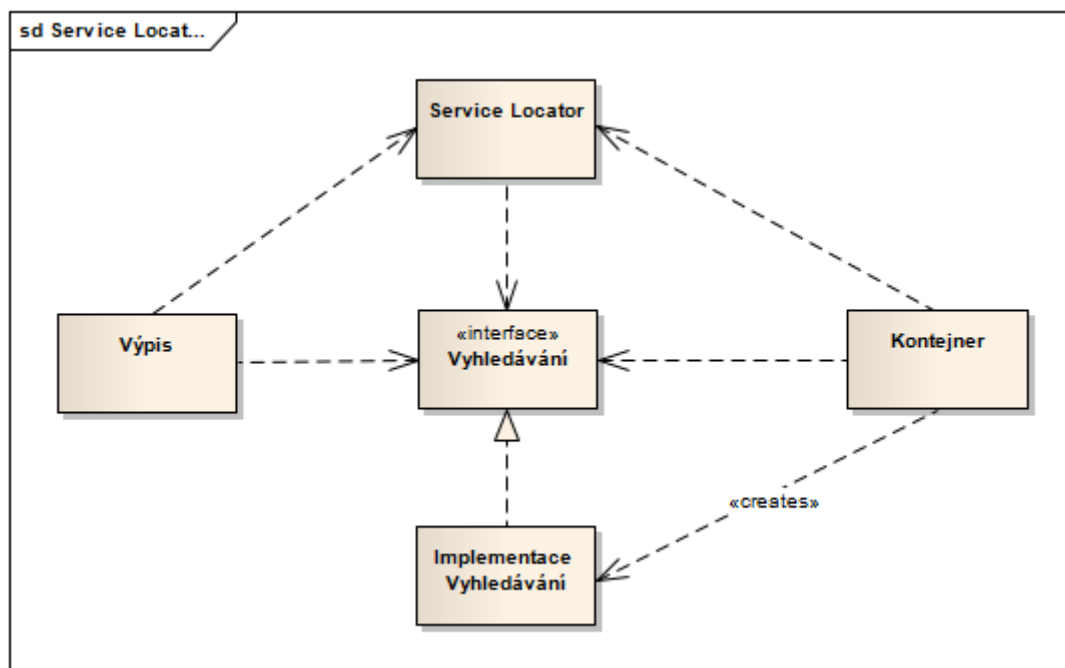
Metody jak dostat závislá rozhraní do třídy jsou tři.

- Constructor injection – parametr v konstruktoru při vytváření instance třídy slouží jako prostor pro Dependency Injection. Nevýhoda je, že závislost lze vyřešit jen při vytváření instance na počátku. Výhodou je, že třída má zaručeno naplnění závislosti.
- Setter injection – vyřešení závislosti pomocí setteru. výhoda je, že se kvůli závislosti nemusí měnit konstruktor třídy. Nevýhoda je zřejmá. Je nutné dávat pozor na včasné naplnění závislosti. Toto se bohužel nepozná při sestavování, ale až při běhu.
- Interface injection – tato metoda není tak obvyklá a spočívá v implementaci injekčních rozhraní obsahujících potřebné informace. Další vlastnosti a detaily této metody jsou uvedeny v článku [11].

Pro tento návrhový vzor existují hotová řešení pro každý objektově orientovaný jazyk v podobě IoC Containerů.

4.7.5.2 Service Locator

Další velmi podobný vzor pro řešení závislostí. Jeho nevýhodou je, že jde hůře testovat unit testy. Následující obrázek 4.8 popisuje způsob chování.



Obrázek 4.8: Závislosti ve vzoru Service Locator

Z diagramu vyplývá podobně jako u vzoru *Dependency Injection*, že má opět vzor objekt **Service Locator**, který se stará o vyhledávání služeb vyžadovaných ve třídě **Výpis** a voláním určité metody dodá třídě **Výpis** implementaci **Implementace Vyhledávání**. Samozřejmě je třeba opět vyřešit závislost při dodávání **Service Locator** do **Výpisu**.

Od vzoru *Service Locator* se v poslední době ustupuje vzhledem k požadavkům na testovatelnost aplikací pomocí unit testů, které se píšou o něco složitěji než pro vzor *Dependency Injection*.

4.8 Architektura

Požadavky na architekturu jsou vzhledem k rozsáhlosti aplikace a její komplikovanosti velmi specifické. Jedná se o webovou aplikaci a tedy můžeme vybrat jako základ některý ze známých architektonických vzorů.

Webové architektury se vytvářejí nejčastěji jako 2 nebo 3 úrovně. Aplikace je tak rozdělena do částí starajících se o

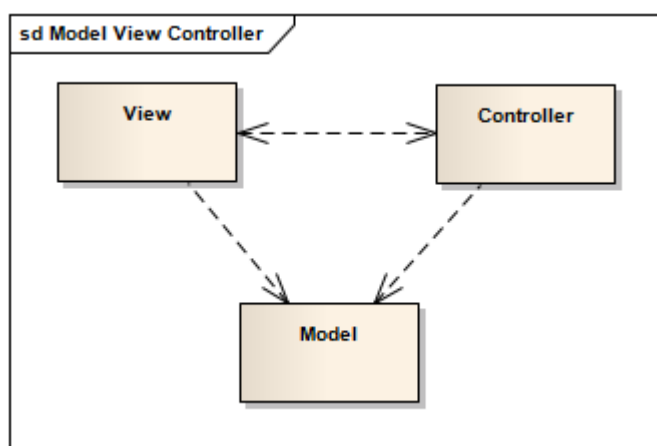
1. Zpracování požadavků uživatele
2. Práci s daty v aplikaci
3. Prezencaci výstupu uživateli

Dvouúrovňová architektura je pouze spojením některých dvou z výše zmíněných částí. O celku nerozlišujícím mezi těmito částmi nelze pak mluvit o architektuře, ale jde o amatérský počín bez smysluplného uspořádání, kterým si většina z programátorů webových aplikací zajisté sama prošla.

Jako nejznámější architektonický vzor využívající všech tří vrstev je dlouho známý vzor Model View Controller. Nemá cenu rozebírat jiné vzory, protože jsou to varianty na tento vzor.

4.8.1 Model View Controller (MVC)

Vzor je známý již ze 70. let minulého století, kdy s ním jako první oficiálně přišel Trygve Reenskaug jako s frameworkem pro SmallTalk. Od té doby sehrál významnou roli v návrhu frameworků a aplikací s uživatelskými rozhraními[5]. Základní princip popisuje velmi jednoduchý diagram 4.9.



Obrázek 4.9: Model View Controller

V této architektuře jsou důležité již dříve zmiňované prvky, tentokrát s konkrétními pojmenováními.

Model se v aplikaci stará o poskytování a změnu dat, ať už z databáze nebo z jiných zdrojů. Jde o objekt představující čistě data bez jakýchkoli vizuálních prvků.

View česky *pohled*, se stará o zobrazení modelu pomocí vizuálních prvků uživateli. Využívá k tomu technologie HTML a dalších doplňků. Data nemá možnost měnit.

Controller česky *řídící část*, starající se o zpracování požadavků přicházejících od uživatele. Po zpracování požadavku změní *model* a nastaví správné zobrazení *pohledu*.

Další podrobnější informace lze najít v [5]. Vzor MVC je implementován v různých frameworkích třetích stran u většiny zmíněných programovacích platformách v sekci 3.

4.8.2 Modularita v Model View Controller

Z pohledu požadavků na aplikaci modulárního e-shopu je nutné uvažovat o vhodné rozšiřitelnosti architektonického vzoru MVC. Nabízí se několik možností:

1. Rozšíření o další Controllery, které budou ovládat rozšiřující funkce. Tato možnost je samozřejmá ve všech frameworkích podporujících MVC. Controllery jsou věnované vždy několika společným funkcím. Například Controller pro zboží, zákazníky, objednávky, přihlašování, atd. V tomto případě využívají Controllery vlastní Pohledy. Modely mohou používat buď společné nebo opět vlastní. V případě kompilačních jazyků se musí vždy aplikace znovu sestavit a spustit, což je nevýhoda.
2. Rozšíření o skupiny Controllerů. Ve frameworkích se těmto skupinám říká *oblasti* (*Areas*) nebo *moduly* (*Modules*). Nejedná se přímo o připravené moduly fungující ihned po připojení k aplikaci. U kompilačních jazyků se musí opět aplikace sestavit a spustit. Rozšiřující oblasti jsou v souborovém systému rozděleny podle adresářů nebo jmenných prostorů (*namespaces*) a často se jedná o oddělení aplikačních částí frontendu a backendu. V prostředí .NET ve frameworku MVC existují rozšíření, která fungují na tomto principu rozšíření. Jedním je například Portable Areas ², projekt se ale dále již nerozvíjí.
3. Rozšíření o externí balíčky obsahující kompletní rozšiřující funkce včetně Controllerů, Pohledů i Modelů. Jde o nejpokročilejší možnost ze zmiňovaných. Vlastnosti předchozích možností jsou využity i zde. Jde ale o způsob nahrávání a integrace těchto funkcí do již hotového řešení. Aplikaci je třeba navrhnout tak, aby poskytovala body pro rozšíření i mimo předchozí možnost seskupování Controllerů. V této variantě jdou rozšířit i další části hotové aplikace včetně Modelů a Pohledů. Nevýhoda tohoto řešení je složitost. Výhoda je, že se hotová aplikace nemusí znovu sestavovat, stačí dodat jen sestavené moduly a aplikaci spustit.

Implementace modulární architektury si vyžádala nejsložitější řešení vzhledem k požadavkům na aplikaci e-shopu. Specifiky tohoto řešení se zabývá kapitola Implementace 5.

4.9 Implementační prostředí

Zvoleným prostředím byla po provedení Rešerše v kapitole 3 zvolena platforma Microsoft .NET. Zvolil jsem tuto platformu z několika důvodů:

- Platforma .NET je navržena a udržována velkou softwarovou společností.
- Platforma má velké množství uživatelů.

²<<http://portableareas.codeplex.com/>>

- Platforma je celistvá a poskytuje vlastní kvalitní IDE - Visual Studio.
- Pracovní nástroje platformy jsou dostupné zdarma.
- Existují hostingy i nabídky dedikovaných serverů pro tuto platformu.
- Platforma se rychle vyvíjí.
- Pro platformu je dostupný framework pro webové aplikace s architekturou MVC v dostatečně odladěné a rozšiřitelné verzi.
- Použitý programovací jazyk C# je podobný jazyku Java, se kterým jsem již pracoval.
- Na této platformě existuje spousta rozšiřujících knihoven ulehčujících práci.
- Na této platformě jsem nikdy neprogramoval webové aplikace, a proto je pro mě tato volba i výzvou.

4.9.1 Rozšiřující knihovny pro platformu .NET

V této části budou probrány zajímavé použité knihovny platformy Microsoft.NET použité v této práci. Některé z těchto knihoven jsou lehce integrovatelné do projektů pomocí speciálního balíčkovacího systému **NuGet**³, který je dostupný jako rozšíření pro Visual Studio. *NuGet* poskytuje rozšíření, která se snadno instalují jedním kliknutím a sám se postará o jejich nastavení. Rozšíření pro systém *NuGet* jsou poskytovány jak nezávislími vývojáři, tak přímo od Microsoftu. Knihovnu pro *NuGet* může napsat každý snadno a rychle.

4.9.1.1 ASP.NET MVC 3

V první řadě je třeba zmínit hlavní knihovnu, na které je jádro aplikace postavené a tou je framework ASP.NET MVC 3. Něco málo o něm bylo řečeno už v kapitole Rešerše 3.2.2 a o jeho architektuře v předchozí sekci 4.8.1. Všechny jeho vlastnosti jsou detailně popsány na webových stránkách frameworku⁴ pomocí skvělých video a textových tutoriálů.

4.9.1.2 Entity Framework - Code First

Microsoft přišel před několika lety s novým ORM frameworkem, který se stal soupeřem NHibernate. V poslední verzi 4 už je Entity Framework⁵ na použitelné úrovni. Entity Framework nabízí 3 různé způsoby, jak založit databázové úložiště.

- Jedním způsobem je *Code First*. Spočívá v nulovém návrhu databáze. Proto se mu říká *Code First (nejdříve kód)*. Nejprve se navrhnou entity ve formě tříd a jejich vztahy se naváží přes otypované parametry. Jakákoli změna entit je detekována a databáze se podle pravidel přeorganizuje. V další verzi 4.2 se představí vlastnost zvaná *Migrations*, která dokáže opravit změny v databázi bez nutnosti přehrávání celé databáze, což byla

³<http://nuget.org/>

⁴<http://www.asp.net/mvc>

⁵<http://msdn.com/data/ef>

velice kritizovaná vlastnost. Musela se řešit zálohováním dat nebo zapsáním pevných iniciálních dat do kódu. Jinak se Microsoft snaží tento způsob návrhu databáze upřednostňovat ve všech svých tutoriálech.

- Další způsob je *Model First*. Jde o způsob, který předpokládá, že má programátor již hotový model databáze ve formátu vhodném pro Entity Framework. Z modelu se vygenerují třídy zastupující entity a dále se pokračuje jako při prvním způsobu.
- Poslední třetí způsob je *Database First*. Tento případ předpokládá, že programátor má přístup k již existující databázi. Z ní se potom vygenerují opět entity.

Model, přes který se převádí databáze na entity a naopak se jmenuje *Entity Data Model* (EDM) a staví na známém E-R modelu od Dr. Petera Chena. K dotazování na databázi se využívá jazyka LINQ, který poskytuje nápovědu při psaní kódu a kontrolu syntaxe během kompilace. Entity Framework je postavený nad ADO.NET provider modelem a proto jsou přes něj dostupné všechny funkce. Díky tomu také se může aplikace připojit na různé databáze od MSSQL přes DB2 po Oracle[14].

4.9.1.3 SQL Compact Edition

Nejedná se o knihovnu, ale o databázi, ale je třeba o ní ztratit pár slov, protože byla použita v aplikaci. Je to nejmenší verze z nabízených databází Microsoftu a je přednastavená při vytváření základní šablony aplikace ve frameworku MVC3 ve Visual Studiu. V projektu jde o verzi SQL Server Compact Edition 4.

Jde o databázi uloženou v jednom souboru společně s aplikací, její možnosti jsou omezené. Nevýhodou je, že nejde profilovat dotazy do databáze pomocí profesionálních databázových nástrojů jako je SQL Server Management Studio. Na druhou stranu je zdarma a obsahuje kompaktní databázový systém přímo v jednom souboru, takže není třeba databázový systém instalovat. Microsoft bohužel nenabízí srovnání s ostatními verzemi, ale našel jsem dokument, který se tímto zabývá⁶. Jinak srovnání a další dostupné verze, mezi nimiž je třeba zmínit edici Express, která je zdarma, je dostupné online na webových stránkách MS SQL Serveru⁷.

4.9.1.4 Castle Windsor

Tato knihovna je jedním z mnoha *Inversion of Control kontejnerů* pro ASP.NET framework. Byla vybrána z důvodu potřeby použít IoC kvůli načítání funkcí z modulů. Pro Castle Windsor hovoří dobrá dokumentace⁸ a množství možností. Důležitá je možnost načítat implementace rozhraní pomocí reflexe ze *sestavení* (*assemblies*). Funguje na principu vzoru *Dependency Injection*. Nejprve se vytvoří *Container*, do něho se registrují rozhraní a jejich implementace. Poté se závislosti při běhu programu řeší na požádání.

⁶<http://coolthingoftheday.blogspot.com/2011/01/sql-server-compact-4-vs-sql-server.html>

⁷<http://www.microsoft.com/sqlserver/en/us/product-info/compare.aspx>

⁸<http://stw.castleproject.org/Windsor.MainPage.ashx>

4.9.2 Paged List

Pro stránkování byla využita knihovna *Paged List*⁹. Její přednosti jsou, že usnadní práci při vytváření odstránkových seznamů a nabízí možnosti nastavení vzhledu stránkovacích odkazů na pohyb mezi stránkami.

4.9.2.1 Glimpse

Tato vynikající knihovna¹⁰ je takový doktor na všechno. Je to diagnostický nástroj zabudovaný přímo do aplikace. Načte se na požádání s rozhraním webové aplikace jako další okno a obsahuje důležité informace, které debugger Visual Studio nezná. Díky němu se dají aplikace napsané v MVC frameworku ladit jednoduše a rychle. Knihovna zamezí spoustu problémům a ušetří pár zničených klávesnic a monitorů u více vznětlivých webových vývojářů.

4.9.3 Ostatní použité technologie

Ještě stojí krátce za zmínku použité technologie na straně uživatelského rozhraní, které jsou standardně použité už v šabloně projektu MVC frameworku ve Visual Studiu.

HTML 5

HTML 5 je novou verzí dlouho ctěného standardu HTML 4, který je pro potřeby moderního webu již zastaralý. 5. verze přináší nové tagy pro audio, video a grafiku. Dále se zaměřuje na vylepšení sémantického významu částí stránky pomocí dalších tagů jako je section, article, header a nav. Další změny jsou ve formulářových polích, které by měly podporovat nové způsoby vkládání a nové datové typy. Další je podpora ukládání dat na straně uživatele a jejich zachovávání¹¹. HTML 5 bude muset urazit ještě dlouhou cestu standardizačními procesy, než se stane obecně standardem. Naštěstí se Microsoft vzpamatoval a poslední verze Internet Exploreru už nestandardizované HTML 5 podporuje.

CSS 3

Další verze kaskádových stylů na formátování vzhledu HTML dokumentů. Ve verzi 3 ještě nejsou také standardizované, ale prohlížeče je částečně podporují, nebo lze využít vlastních tagů prohlížečů. Poslední verze nabízí vychytávky jako víceobrázková pozadí, přechody, transparentnost, více-sloupcový layout, média, atd. Bohužel si ještě nějaký rok budeme muset na finální standard počkat.

⁹<https://github.com/TroyGoode/PagedList>

¹⁰<http://getglimpse.com/>

¹¹<http://en.wikipedia.org/wiki/HTML5>

Javascript a JQuery

Javascript¹² tu s námi je už od roku 1997, kdy byl standardizován asociací ECMA. Jde o netypový objektový skriptovací jazyk používaný pro manipulaci se vzhledem a daty na straně uživatele. Přestože je velmi náchylný k chybám, zažil boom s nástupem AJAXu a frameworků jako je *JQuery*¹³, které usnadňují skriptování a nabízejí přidané funkce. Díky JQuery bylo napsáno mnoho pluginů, které vylepšují přívětivost webů a usnadňují navigaci. V aplikaci e-shopu je například použit jako standardní validátor uživatelských formulářů JQuery plugin *Validation*¹⁴, který spolupracuje s ASP.NET MVC 3 frameworkem a předává si informace o formulářových políčkách a jejich obsahu.

Dokonce i javascriptové knihovny jako je *JQuery* a její pluginy jsou vhodné pro distribuci pomocí balíčkovacího systému *NuGet*, který je zmíněn v úvodu této sekce o Rozšíření 4.9.1.

Ještě k doplnění, experimentální uživatelské rozhraní aplikace využívá také *JQuery* pluginy *JQuery UI*¹⁵, *Modernizr*¹⁶ a *JsTree*¹⁷.

¹²<<http://cs.wikipedia.org/wiki/JavaScript>>

¹³<<http://jquery.com>>

¹⁴<<http://bassistance.de/jquery-plugins/jquery-plugin-validation/>>

¹⁵<<http://jqueryui.com/>>

¹⁶<<http://www.modernizr.com/>>

¹⁷<<http://www.jstree.com/>>

Kapitola 5

Realizace

Tato kapitola se věnuje rozboru implementace e-shopu. Kromě popisu nestandardních postupů v implementaci také okrajově popisuje standardní způsoby realizace.

Znázornění struktury aplikace podle komponentů v balíčcích je zobrazeno v diagramu 5.1. Popis částí komponent následuje v další sekci.

5.1 Komponenty aplikace

Z důvodu velké složitosti struktury aplikace je zde popsán diagram komponent, který graficky znázorňuje části aplikace v balíčcích a v nich obsažených komponentách, viz Obrázky 5.1 a 5.2.

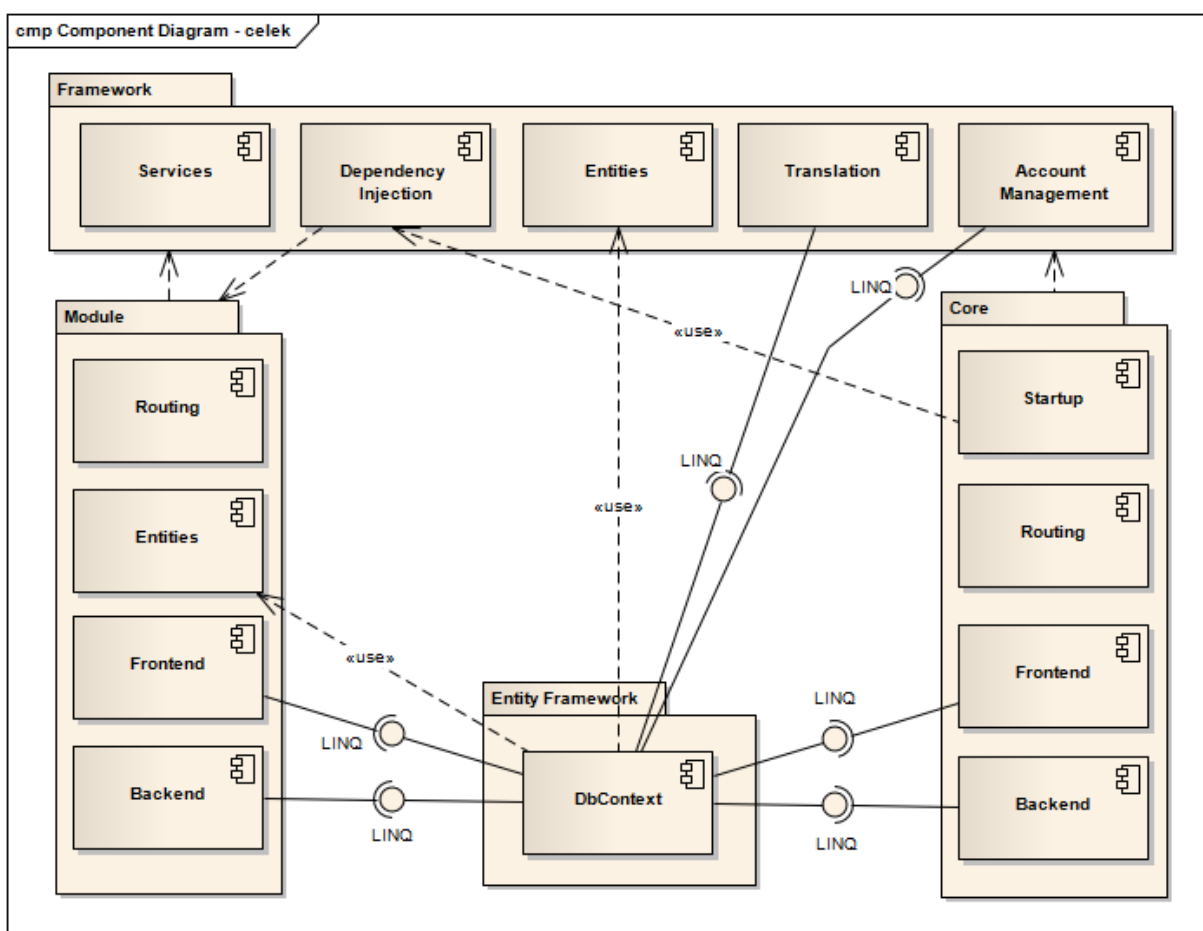
První schéma 5.1 popisuje závislosti mezi sestavením frameworku, jádra a modulů. Všechny tyto části se napojují na Entity Framework pro získání dat z databáze pomocí dotazů v jazyku LINQ. Sestavení Framework obsahuje rozhraní služeb – Services, datové entity – Entities, jazykový překladač – Translator, správu účtů – Account Management a komponentu pro řešení závislostí – Dependency Injection.

Na sestavení Framework závisí pak sestavení jádra – Core a všechna sestavení modulů Module. Tyto sestavení pak mají další komponenty. Sestavení Core obsahuje kromě společných komponent s Moduly (Frontend, Backend, Routing) navíc komponentu Startup, která se stará o inicializaci a spuštění aplikace a používá i funkce komponenty Dependency Injection v sestavení Framework, aby vyřešila závislosti při načítání komponent ze sestavení všech přidáných Modulů do základu aplikace.

Obrázek 5.1 obsahuje pouze nejdůležitější vazby mezi sestaveními a komponentami. Je to z důvodu přehlednosti. Další obrázek 5.2 zobrazuje vnořené komponenty v balíčcích Frontend a Backend. Balíčky jsou v diagramu 5.1 zobrazeny jako komponenty v sestavení Core a Module. Obsahují standardní komponenty MVC architektury, které jsou v balíčku Frontend doplněné o komponentu Widgets. Tu si probereme v sekci 5.4.5.7.

5.2 Použité části ASP.NET MVC 3

V této části si probereme základní použité části frameworku ASP.NET MVC 3, které byly využity v aplikaci. Tyto části zde zmíníme do hloubky nutné k pochopení dalšího popisu



Obrázek 5.1: Diagram komponent

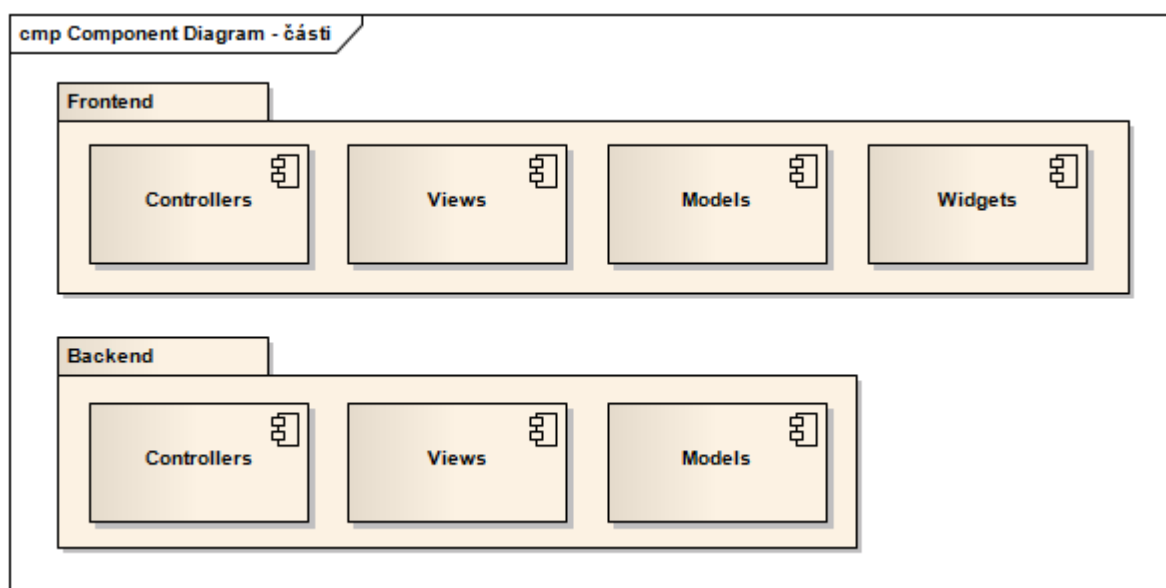
aplikace. Doporučuji však pro bližší seznámení s frameworkem také tutoriály ¹ a knihy [6] a [26].

Pro přehlednost stojí za zmínku základní adresářová struktura projektu používající framework MVC 3, viz. Obrázek 5.3. Povinné v ní není prakticky nic, ale pokud bychom se rozhodli strukturu nedodržet, musíme přinejmenším upravit cesty pro Views specifikací `RazorViewEngine`, o tom dále v 5.2.3. Properties obsahují konfiguraci projektu. References je pseudoadresář pro zobrazení načtených potřebných dynamických knihoven.

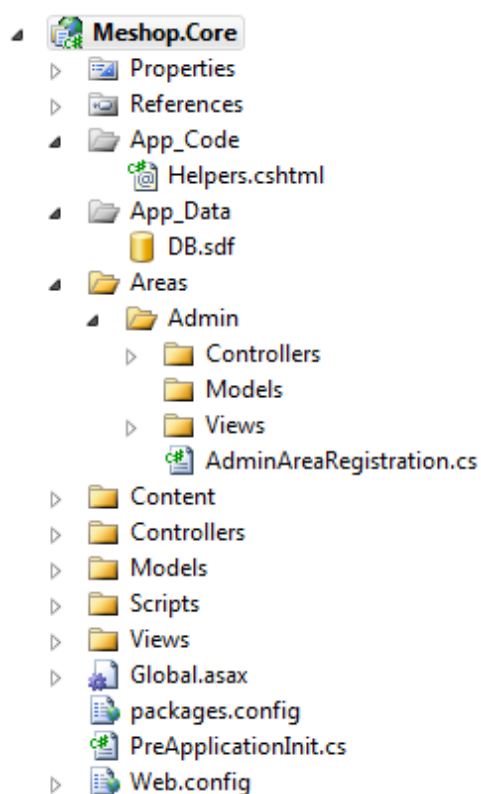
V adresářích s předponou `App_` jsou zdroje používané pro databázové soubory, textové prostředky – `resources` a sdílené třídy. Adresář `Areas` obsahuje *oblasti* oddělené od sdílených tříd v projektu pomocí `namespace`. Adresáře `Content` a `Scripts` obsahují soubory pro zobrazení a client-side skripty.

Ostatní adresáře jsou jasné podle jejich pojmenování a obsahují třídy a prvky popsané níže. Soubor `Global.asax` je startovním souborem zastupující modul `Startup`. Nepovinný

¹<http://www.asp.net/mvc/tutorials>



Obrázek 5.2: Diagram komponent – část nižší úrovně



Obrázek 5.3: Základní struktura projektu

soubor `PreApplicationInit.cs` obsahuje prvky pro inicializaci sestavení modulů při startu aplikace. Soubor `Web.config` je XML konfigurace aplikace, kterou používá IIS web server. `Packages.config` je XML soubor pro systém *NuGet* balíčků s výpisem instalovaných knihoven z *Nuget* repozitáře.

5.2.1 Global.asax

Soubor `Global.asax` se spouští jako první při startu aplikace, proto je vhodné ho zmínit na začátku. Obsahují ho všechny aplikace v ASP.NET. Jako ukázka poslouží následující zhuštěný a ořezaný kód 5.1 přímo z aplikace.

```
1 public class MvcApplication : HttpApplication
2 {
3     protected void Application_Start()
4     {
5         Framework.DI.Modules.Initialize();
6         RegisterRoutes(RouteTable.Routes);
7         Framework.DI.Modules.RegisterAdapters();
8         ControllerBuilder.Current.SetControllerFactory(
9             Framework.DI.Modules.GetControllerFactory());
10        ViewEngines.Engines.Add(new MeshopViewEngine());
11        //database refill
12        Database.SetInitializer(new DatabaseInitializer());
13    }
14    protected void Application_End()
15    {
16        Framework.DI.Modules.Dispose();
17    }
18 }
```

Zdrojový kód 5.1: Global.asax

V metodě `Application_start` se registrují jednotlivé části frameworku MVC a komponenty vlastního frameworku aplikace, které mění standardní funkčnost. Detaily jednotlivých částí jsou probrány v následujících podsekcích, v další sekci rozšíření 5.3 a připojení modulů 5.4.4. Po ukončení aplikace, což probíhá u webových aplikací zřídka, se spustí metoda `Application_End`, která zavolá v tomto případě metodu pro uvolnění registrovaných dat v IoC kontejneru z paměti.

5.2.2 Controller

`Controller`² je abstraktní třída, která poskytuje metody odpovídající na HTTP požadavky, viz. architektura 4.8.1. Specifikací této třídy implementujeme vlastní metody, které poskytují uživateli obsah z Modelů pomocí pohledů – View.

V ukázce kódu 5.2 je vidět jak se provádí specifikace. Metoda `Index` vrací výsledek `ActionResult` pomocí metody `View` s parametrem dat z modelu. Framework MVC zpracuje tento výsledek pomocí vnitřních mechanismů a odešle zpracovaný pohled.

²<http://msdn.microsoft.com/en-us/library/system.web.mvc.controller.aspx>

```

1 public class HomeController : Controller {
2     public ActionResult Index()
3     {
4         var products = ProductModel.GetAll();
5         return View(products);
6     }
7     // .. dalsi metody ...
8 }

```

Zdrojový kód 5.2: Controller

```

1 @model IEnumerable<Meshop.Framework.Model.BasicCategory>
2 @{
3     ViewBag.Title = "Categories";
4 }
5 <h2>@ViewBag.Title</h2>
6 <h3>@T("About")</h3>
7
8 @Html.ActionLink("Edit", "Edit", null, new { id = "edit" })
9
10 <div id="categories">
11     <ul>
12         @foreach (var item in Model) {
13             <li>First</li>
14             @Helpers.RecurseBasicCategory(item, Html)
15         }
16     </ul>
17 </div>

```

Zdrojový kód 5.3: Pohled s jazykem Razor

5.2.3 View

View neboli Pohled se používá pro zobrazení odpovědi aplikace uživateli, viz. architektura MVC 4.8.1. V případě frameworku MVC jde o několikavrstvého poskytovatele pohledů – **RazorViewEngine**³. Jako obsah se používají soubory, které implicitně dědí od **WebViewPage**⁴. Obsahem jsou šablony v šablonovacím jazyku Razor, který je rychlý a intuitivní pro vývojáře, kteří už nějaký šablonovací jazyk používali. Kód 5.3 poskytuje malou ukázkou.

Jazyk Razor se vepisuje přímo do HTML tagů a kombinuje se s nimi. Ke slovu také přichází tzv. **Helper**y, které usnadňují psaní a generují kód ve spojení s HTML. Proces od napsání šablony po zobrazení je ve frameworku několikafázový. Nejprve se napíše pohled a propojí se s **Controller**em, poté se aplikace zkompile, ale pohledy standardně zůstanou nezkompileované a nahrají se do ostrého prostředí ve struktuře shodné s vývojovou. Až teprve při požadavku na zobrazení pohledu se pohled dynamicky na požádání přeloží do C# jazyka, zkompile se a je poslán jako odpověď přes HTTP k uživateli.

³<http://msdn.microsoft.com/en-us/library/system.web.mvc.razorviewengine.aspx>

⁴<http://msdn.microsoft.com/en-us/library/system.web.mvc.webviewpage.aspx>

Za zmínku stojí silně a dynamicky typované proměnné – *silně typovaný Model* a *dynamický ViewBag*, které předávají data z akcí *Controlleru*. Systém pohledů také umožňuje sdílet *Layout* pro skupiny pohledů.

Pokud se vrátíme k ukázce 5.3, můžeme si všimnout použití výše zmíněných prvků. Silně otypovaný model obsahující list entit na řádku 1. Přepnutí do jazyka *Razor* pomocí `@` na řádku 2. Použití dynamické sdílené proměnné *ViewBag* na řádku 3. Ta je i následně na řádku 5 použita pro výpis. Na řádku 6 je použita vlastní implementace překládání textu, o které si řekneme více v sekci 5.3.2. Snad jen zmíním část implementace v pohledech. T je vlastní parametr ve vlastní verzi rodičovské třídy *WebViewPage* plněný singletonem *TranslationResolver*. Na řádku 8 je ukázka standardního helperu MVC. 12. řádek pak ukazuje možnosti iterace v šablonách a kombinaci HTML s *Razor*. 14. řádek je ukázka volání vlastního *Helperu*.

5.2.4 Model

Modely jsou ve frameworku MVC celkem komplikované, protože data reprezentují i sbírají od uživatelů z formulářů a přes *Entity Framework* (v našem případě) je ukládají do nebo načítají z databáze. Pro jejich úplné pochopení doporučuji prostudování kapitol 4,6 a 13 z knihy [6].

Ve zkratce jsou modely sady entit pro práci se specifickými pohledy a nemusí odpovídat tabulkám v databázi. Modelem může být entita nebo třída obsahující entity a parametry. V předchozí ukázce kódu pohledu 5.3 je modelem list entit, které se podle požadavků vypíší na obrazovku do formuláře nebo jako text. V následujícím kódu 5.4 je ukázka modelu obsahující entity pro zobrazení v Pohledu.

V další ukázce 5.5 je kód entity sloužící zároveň i jako model. Je zde zároveň vidět, jaké entita používá doplňkové atributy pro specifikaci názvu pole, který je možné přeložit – atribut *TranslateName*, dále jsou zde atributy pro upřesnění *Entity Frameworku* jaké má aplikovat omezení a klíče v DDL pro databázi. Více o attributech v sekci Validace dat 5.2.8 a Atributy 5.2.9.

```

1 public class CategoryModel
2 {
3     public BasicCategory Category { get; set; }
4     public IEnumerable<BasicCategory> Categories { get; set; }
5     public IEnumerable<BasicProduct> Products { get; set; }
6
7     public CategoryModel(BasicCategory c, IEnumerable<BasicProduct> p, ←
8         IEnumerable<BasicCategory> cs )
9     {
10         Categories = cs;
11         Category = c;
12         Products = p;
13     }
14 }

```

Zdrojový kód 5.4: Model z více entit

```
1 public class BasicProduct
2 {
3     [Key]
4     [ScaffoldColumn(false)]
5     public int ProductID { get; set; }
6
7     [Required]
8     [TranslateName("Name")]
9     public string Name { get; set; }
10
11     [Required]
12     [TranslateName("Price")]
13     public decimal Price { get; set; }
14
15     [TranslateName("Categories")]
16     public List<BasicCategory> Categories { get; set; }
17 }
```

Zdrojový kód 5.5: Entita a zároveň Model

5.2.5 Routování

Routování je neoddělitelnou součástí všech moderních frameworků založených na architektuře MVC. Díky této další vlastnosti frameworku jsou adresy URL nasměrovatelné na metody v **Controllerech**. Díky tomu lze vytvářet různé aliasy a textové odkazy vhodné pro SEO i pro čitelnost člověkem. **Route**, nebo-li *cesta* je uložena do datového slovníku a je dostupná obousměrně, pro generování odkazů v šablonách i pro rozpoznání metody v **controlleru** a parametrů podle URL. Z ukázky kódu 5.6 lze snadno vyvodit způsob zápisu cesty. Routy jsou dostupné v aplikaci MVC přes singleton **RouteTable** a jeho metodu **Routes**.

```
1 routes.MapRoute(
2     "Default", // Route name
3     "{controller}/{action}/{id}", // URL with parameters
4     new {
5         controller = "Home",
6         action = "Index",
7         id = UrlParameter.Optional } // Parameter defaults
8 );
```

Zdrojový kód 5.6: Route

Routy se registrují už při startu aplikace, proto je třeba inicializovat registraci v souboru `global.asax`, zmíněném na začátku sekce 5.1.

5.2.6 Areas

Pomocí **Areas**, česky *oblastí*, lze oddělit strukturálně komponenty aplikace. Slouží také pro udržení přehlednosti a zamezení konfliktů se stejnými názvy tříd protože jsou v jiném *namespace*. Jejich registrace se provádí pouze do komponenty **Routes**. Konfigurace je uložena v souboru `AdminAreaRegistration.cs` – předpona je podle oblasti ve které se soubor nachází, viz. Obrázek 5.3. *Oblast* obsahuje všechny základní prvky MVC architektury, stejně jako *root* adresář.

5.2.7 Filtry

Filtry ve frameworku slouží pro spouštění funkcí před nebo po *akcích* v *controllerech*. Dělí se na lokální – přímo napsané v controlleru a globální – aplikované pomocí registrace **Filter Atributů**. Více o filtrech se lze dozvědět v kapitole 13 knihy [26].

Ukázka lokálního action filtru 5.7 ukazuje příklad, jak lze tuto vlastnost využít pro zobrazování košíku, nastavení aplikace a uživatelské role ve všech *pohledech*.

```

1 protected override void OnActionExecuting(ActionExecutingContext ctx)
2 {
3     base.OnActionExecuting(ctx);
4     _cartService.StartCart(ctx.HttpContext);
5
6     var account = new AccountManagement();
7     ViewBag.UserRole = account.IsInRole(User.Identity.Name, "Customer") ? "↔
8     Customer" : "none";
9     ViewBag.Settings = _commonService.Settings;
10 }

```

Zdrojový kód 5.7: Action Filter

5.2.8 Validace dat

Validace dat slouží ke kontrole dat při přijímání formulářů. Ve verzi MVC 3 je dokonce validace prováděna už při vyplňování u uživatele v prohlížeči pomocí pluginu *jQuery Validate*. Funguje dokonce i vzdálená kontrola například zda je přezdívka už obsazená. Validace dat se ve frameworku MVC zapisuje do modelů pomocí atributů, viz. kód entity 5.5. Jsou zde vidět validační atributy typu **Required** – *vyžadováno*. Validace je poskytována přes třídy **ModelValidationProvider** a **ValidationAttribute** které se dají rozšiřovat a tím se zabývají kapitoly 6 a 13 knihy [6].

5.2.9 Atributy

Atributy (jiné programovací jazyky je nazývají *Anotace*) jsou metadata funkcí a tříd. Jsou získávané díky schopnosti reflexe v prostředí .NET. Umožňují tak spoustu věcí zjednodušit. Existuje spousta připravených atributů pro framework MVC, není však vůbec těžké napsat si vlastní. Příkladem je kód Entity 5.5, kde jsou atributy před všemi proměnnými v hranatých závorkách. Další atributy se používají například na Autorizaci a Autentizaci.

5.2.10 Autorizace

Standardní autorizace je ve frameworku MVC řešena použitím hotové implementace rozhraní `MembershipProvider`. Je to velmi rozsáhlá implementace, ale každý si může napsat vlastní. Nepříjemné je, že `MembershipProvider` si standardně vytváří v databázi vlastní tabulky a nespolupracuje s *Entity Frameworkem*. To způsobuje konflikty, protože *Entity Framework* detekuje všechny změny ve struktuře. Proto je vhodnější napsat si vlastní implementaci, nebo vyřešit autorizaci vlastním způsobem, což se ale nedoporučuje z důvodu vzniku nových bezpečnostních děr.

Standardní autorizace se dělí na Windows a Forms autentizaci, v běžných aplikacích je vhodná spíše Forms autentizace, protože se při ní *nemusí* vytvářet uživatelské účty Windows. Autorizace je udržována přes statickou třídu `FormsAuthentication` pomocí Cookies, které ukládá u uživatele v prohlížeči.

Pro autentizaci je vhodné mít vyhrazený extra `controller` propagující komponentu autentizace přes `pohledy` k uživateli. Jde vlastně o roli `modelu`. Autorizaci je poté třeba zavést do aplikace pomocí nastavení v hlavním konfiguračním souboru `Web.config`. Více lze najít v obou knihách [6] [26] i v tutoriálech⁵ a detaily v dokumentaci na MSDN⁶.

5.2.11 Autentizace

Autentizace je složitější nadstavba autorizace. Stará se o řízení přístupu uživatelů k prostředkům. Kontroluje oprávnění uživatelů podle jejich rolí a práv. V ASP.NET je opět standardní implementovaná cesta, která nemusí vyhovovat všem ze stejných důvodů jako v případě autorizace.

Při nastavení autentizace jsou potřeba vhodné třídy zapsat do konfiguračního souboru `Web.config`. Jde o implementace tříd `RoleProvider` a `ProfileProvider`. Více se lze o způsobech rozšíření i nasazení hotových řešení lze dočíst opět v knihách [6] a [26] a na MSDN⁷⁸.

5.3 Rozšiřující úpravy jádra

V této části si probereme nutná rozšíření základu aplikace jako jsou spolupráce modelů s databází, návrh frontendu a backendu, uživatelů, lokalizaci, helpery a základní datové entity.

5.3.1 Propojení modelů s databází

Již dříve bylo zmíněno, že se modely připojují k databázi přes *objektově-relační mapování*, které zastupuje v Entity Frameworku *Entity Data Model*. Entity Framework kopíruje mechanismus připojení z nižší vrstvy ADO.NET a staví na něm. Jedná se o třídu `DbContext`, která specifikuje `ObjectContext` a zpřístupňuje mapování. Ukázka kódu 5.8 popisuje část nutnou k propojení s databází způsobem *Code First*. Deklarací *properties* generického typu `DbSet<>` říkáme, které entity chceme namapovat do databáze jako tabulky. Specialitou je

⁵<http://www.asp.net/mvc/tutorials>

⁶<http://msdn.microsoft.com/en-us/library/6tc47t75.aspx>

⁷<http://msdn.microsoft.com/en-us/library/317sza4k.aspx>

⁸<http://msdn.microsoft.com/en-us/library/ta63b872.aspx>

přepsání tohoto typu s parametrem dědicím od třídy `PluginEntity`, který slouží jako místo prostředek pro práci se zásuvnými entitami.

Metoda `OnModelCreating` vkládá prostor pro konfiguraci Entity Frameworku pomocí odebrání defaultních nastavení. Entity Framework totiž ctí heslo „konvence nad konfigurací“, což znamená, že sám od sebe funguje a pouze v případě potřeby změny je možné nastavení odebrat – což ukazuje řádek **17**. Další pokračování metody je popsáno dále v [5.4.5.8](#), protože obsahuje část pro připojení entit z modulů.

```

1 public class DatabaseConnection2 : DbContext
2 {
3     public DbSet<Resource> Resources { set; get; }
4     public DbSet<Setting> Settings { get; set; }
5     public DbSet<Customer> Customers { set; get; }
6     public DbSet<BasicProduct> Products { get; set; }
7     public DbSet<BasicCategory> Categories { get; set; }
8     public DbSet<CartItem> Carts { get; set; }
9     public DbSet<Order> Orders { get; set; }
10    public DbSet<OrderDetail> OrderDetails { get; set; }
11    public new DbSet<TEntity> Set<TEntity>() where TEntity : PluginEntity
12    {
13        return base.Set<TEntity>();
14    }
15    protected override void OnModelCreating(DbModelBuilder modelBuilder)
16    {
17        modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
18        // ... pokračování ...
19    }
20 }

```

Zdrojový kód 5.8: Specifikace třídy `DbContext`

5.3.2 Lokalizace

Standardně se v aplikaci používají statické zdroje, které už uživatel aplikace nemůže dále jednoduše upravovat, proto byl z důvodu větší flexibility a možnosti přidání více jazyků do hotové aplikace předělán celý systém vyhledávání překladů řetězců – tzv. *Resources*. Všechny základní aplikace nebo rady na stackoverflow.com řeší lokalizaci pouze pomocí standardní cesty. Aplikace ale potřebuje mít uložené řetězce v databázi a načítat si je do vlastní aplikační cache, aby byly omezeny dotazy na databázi. Jediné možné řešení jsem našel na webu MSDN [\[3\]](#) pro starou verzi ASP.NET frameworku, která naštěstí ale funguje i v poslední verzi 4.0.

Načítání `DBResourceProvider` se provádí v konfiguračním souboru `web.config` hlavního sestavení Core tagem [5.9](#) s nastavením `Factory`. V tomto nastavení pozná aplikace podle nastaveného jazyka v prohlížeči návštěvníka, jaký má použít překlad.

Ostatní implementace týkající se překládání se nachází v souborech v adresáři `Translation` v sestavení `Framework`. Jde o třídy popisované v tutorialu MSDN [\[3\]](#). Další implementace se týká pohledů a popisků v entitách a stavových zpráv v aplikaci.


```

1 <system.web>
2   <globalization uiCulture="auto" culture="auto"
3     resourceProviderFactoryType="Meshop.Framework.Translation.↔
        DBResourceProviderFactory , Meshop.Framework" />
4   ...dalsi tagy...
5 </system.web>

```

Zdrojový kód 5.9: Konfigurace DBResourceProvider v lokalizaci

Problémem bylo použití lokalizačních atributů pro entity, které jsou kontrolovány při startu aplikace Entity Frameworkem. Ten atributy entit rovnou načítá, pokud se databáze musí vytvářet znova. Proto byla zavedena ještě jedna třída specifikující `DbContext`, kterou používají `Resources` jen pro sebe. To ale úplně nezamezilo pádům aplikace, proto byla ve třídě `DBResourcesModel` do těla metody `GetResourceByCultureAndKey` načítající překlady pomocí Entity Frameworku přidána klauzule *try-catch* pro odchycení výjimky způsobující pád. Entity tak dostanou původní pojmenování přímo z kódu. To se ale přeloží při použití entit ve formulářích, které jsou generovány podle atributů entit – tzv. *scaffolding*.

Entity využívají následující lokalizační atributy:

- `TranslateName` – překlad názvu atributu
- `TranslateRequired` – překlad hlášení o požadovaném vyplnění
- `TranslateRegularExpression` – překlad hlášení o platnosti pole podle testu na regulární výraz

Ty jsou do spouštěné aplikace registrovány pomocí adaptérů v souboru `Global.asax`. Standardních atributů frameworku .NET využívaných v Entity Frameworku je ovšem více, tyto představují pouze *proof-of-concept*, že i texty v attributech lze překládat. Důležité také je, že atributy jsou překládány i pro entity v modulech. Více o attributech použitelných pro entity a modely na MSDN⁹. Tyto atributy jsou ale standardní a je třeba je podle již hotových přepsat, aby překládaly chybové texty.

Do pohledů byly překlady implementovány díky přepsání rodiče všech pohledů `WebViewPage` vlastní třídou `BaseWebViewPage` obsahující *property* `T` která má jako datový typ delegáta, proto se nemusí používat statická třída způsobem `@Translator.Translate("text")`, ale stačí použít krátkou verzi `@T("text")`. Podobně je použit *Gettext* překlad v jazyku PHP. Delegáty prakticky vysvětluje například MSDN tutoriál¹⁰ nebo videotutorial pro studenty zdarma (registrace na Dreamspark¹¹ nutná) dostupný na Pluralsight¹². Konkrétní metoda volající překladač z přepsaných `Resources` je dosazena do *property* v `BaseWebViewPage` pomocí singletonu `TranslationResolver`.

⁹<http://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations.aspx>

¹⁰<http://msdn.microsoft.com/en-us/library/aa288459.aspx>

¹¹<http://www.dreamspark.com>

¹²<http://www.pluralsight-training.net/microsoft/Courses/TableOfContents?courseName=csharp-fundamentals>

5.3.3 Frontend

Frontend je v balíčku Core implementovaný pomocí controllerů a pohledů. Obsahuje základ potřebný pro splnění triviálních Use Case 4.5. Případy užití jsou dále rozšiřitelné pomocí modulů a změnou nezkompilovaných pohledů, což se hodí pro změnu vzhledu a rozložení layoutu.

V základu frontend obsahuje tyto controllery:

1. `AccountController` – obstarává uživatelské záležitosti jako je přihlašování a registrace.
2. `HomeController` – poskytuje zobrazení vstupní stránky, košíku, zboží a katalogu.
3. `CheckoutController` – obsluhuje všechny aktivity spojené s nákupem zboží. Tato část je implementována podle tutoriálu MVC Music Store [8].

5.3.4 Backend – administrace

Backend je řešený jako oddělená část pomocí `Areas`, což je vidět i na obrázku 5.3. Na rozdíl od běžné části se oblasti musí registrovat, což je popsáno v popisu oblastí 5.2.6. Dále je třeba zajistit oblast před nepovolanými osobami, proto je zde zavedena autentizace. Zabezpečení se provádí pomocí vlastního atributu `Admin` nebo děděním třídy `AdminController`, která poskytuje základní služby s přístupem k databázi. Dědit třídu v modulech ale znemožňuje dědění dále zmíněné třídy `PluginController`, která zajišťuje funkčnost controllerů v modulech.

Backend v základu disponuje těmito controllery:

- `AdminHomeController` – uvodní controller.
- `CategoriesController` – controller pro správu katalogu.
- `OrdersController` – poskytuje správu objednávek.
- `ProductsController` – slouží pro správu produktů.
- `SettingsController` – obsahuje základní nastavení aplikace.

5.3.5 Uživatelé

Aplikace umožňuje registraci a správu účtu uživatelům. Přihlašování je implementováno podle tutoriálu MVC Music Store [8] pomocí uživatelských rolí, které jsou v základní aplikaci jen `Admin` a `Customer`. Odlišuje se tak správce a zákazníci. Tento test se provádí přímo v atributu `AdminAttribute` a také v implementaci služby `IFront`. Služby si probereme v části modulů 5.4.5.5.

5.3.6 Atributy

Jak už bylo zmíněno v sekci 5.2.9 o attributech frameworku, jde o metadata. Skvělé je, že jako ostatní části .NET se atributy v aplikaci dají rozšířit a umožňují tak několik funkcí, na které nejsou ale nijak limitované:

- **AdminAttribute** – již výše zmíněný atribut pro zabezpečení backendu. Dědí od **AuthorizeAttribute**.
- **TranslateName** a další – atributy sloužící k překladu názvů položek a stavových hlášení v entitách.
- **MenuAttribute** – atribut sloužící k rozšíření hlavních menu ve frontendu a backendu položkami z controllerů v modulech. Je použit i v balíčku **Core** pro sjednocení funkcí. Více informací následuje v sekci 5.4.5.6.
- **PlacementAttribute** – slouží k určení místa a události, kde se má zobrazit **Widget**, více o **Widgetech** v sekci 5.4.5.7.

5.4 Moduly

Moduly jsou v aplikaci řešeny jako balíčky ze samostatných projektů. Projekty jsou kompilovány do dynamických knihoven a spolu s nezkompilovanými pohledy a konfiguračními soubory jsou nahrány do určeného adresáře jádra aplikace.

5.4.1 Start aplikace

Start aplikace je z pohledu modulů jednoduchý. Nejprve se při startu aplikace spouští vlastní inicializační funkce **PreApplicationInit** ve třídě označené atributem **PreApplicationStartMethod**. Tato funkce je umístěna v souboru **PreApplicationInit.cs** v balíčku **Core**. Jako další se provede již zmiňovaná funkce **Application_Start** v souboru **Global.asax** v tomtéž balíčku. V těchto funkcích je proto třeba umístit všechny načítací mechanismy. Tomu se věnují další podsekcce.

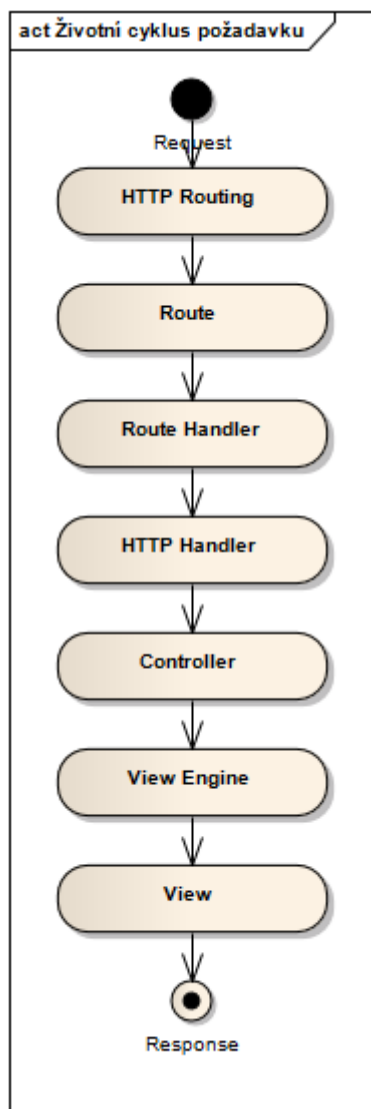
5.4.2 Životní cyklus aplikace

Pro pochopení modulární části je vhodné projít si způsob jakým je aplikace ve frameworku MVC spouštěna, a jaký je životní cyklus vyřizování požadavků. Bohužel jsem se nedopátral žádného oficiálního popisu životního cyklu požadavků a musím zde použít životní cyklus 5.4 zmíněný na [stackoverflow.com](http://stackoverflow.com/a/460165/733748)¹³, který doplním vlastními zkušenostmi při průzkumu spouštěcích sekvencí ve vlastní aplikaci.

Životní cyklus požadavků je zde velmi stručně popsán. Nejprve zjistí podle svého routovacího slovníku, která *routa* platí. Poté se podle routy ve slovníku spustí **controller** a jeho *metoda*. Ještě před spuštěním dané metody se provedou metody *filtrů*. Filtry jsou ostatně

¹³<<http://stackoverflow.com/a/460165/733748>>

nastavitelné na spouštění v jakémkoli stádiu spouštění metody controlleru. Po ukončení metody se výsledek předá *pohledu*, který se zkompileje, spustí a výsledek, nejčastěji ve formátu HTML, se odešle uživateli.



Obrázek 5.4: Životní cyklus požadavku

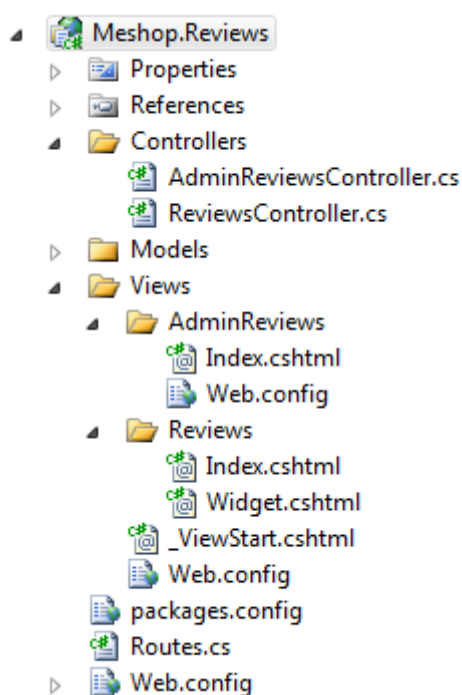
5.4.3 Struktura modulu

Každý modul musí mít danou strukturu, aby mohl být načten jádrem a správně fungovat. Na obrázku 5.5 je struktura vzorového modulu. Projekt je založený na standardní šabloně ASP.NET MVC 3 Web Application. Je třeba ji ale upravit aby se nespouštěla samostatně, ale kompilovala se do adresáře `extension/plugins` v hlavním projektu `Core`. Toto se nastavuje v `Properties - Build - Output Path`. V `Properties` je třeba nastavit také záložku `Web`.

Musíme tam nastavit volbu `Use Custom Web Server` na vymyšlenou hodnotu. Konečně se musí změnit nastavení v `Properties` pod `Solution Explorer` (kliknutím na položku projektu) v položce `Always Start When Debugging` na `False`.

Také se musí do `References` přidat odkaz na projekt `Framework`, aby se mohly využívat jeho třídy a rozhraní. Dále se musí všechny soubory, které se nekompilují (zpravidla `Web.config`, `*.cshtml`, `javascript`, `CSS` a `média`), ve vlastnostech souboru změnit v položce `Copy to Output Directory` na `Copy Always`, protože by se jinak samy k aplikaci nepřidaly.

Další nastavení projektu budou ještě postupně přidávány, ale ve stručnosti jsou upraveny všechny soubory `Web.config`, `_ViewStart.cshtml` a nový je soubor `Routes.cs`. Změny v souborech oproti standardnímu nastavení budou dále popsány. Soubory `Web.config` jsou v modulech jen pro přepsání rodiče pohledů a na import základních *assemblies* a *namespaces*.



Obrázek 5.5: Struktura projektu modulu

5.4.4 Připojení modulů

Připojování modulů je v aplikaci poloautomatické. Ve vývojovém režimu jde zcela automatizovat již výše zmíněným upravením nastavení při kompilování a spouštění přímo v adresáři celého balíku projektů – *Solution*. To probíhá automaticky při spouštění při vývoji přímo z Visual Studia na vývojovém serveru IIS Express.

Po dlouhém hledání se mi podařilo najít článek Shannona Deminicka [4], který se zabývá připojováním sestavení do jiných sestavení až po zkompilování. Proto jsem použil postup v něm navržený. Jde o systematické prohledávání daných adresářů a načítání tříd z nalezených

sestavení ještě než se spustí samotná aplikace. K tomu slouží soubor `PreApplicationInit.cs` popsany výše v sekci 5.4.1. vnitřní metoda celé moduly kopíruje do spouštěcího dočasněho adresáře, protože se v původním adresáři zamknou a nejdou otevřít. Zároveň je třeba nastavit v balíčku `Core` v souboru `Web.config` v tagu `probing` atribut `privatePath` na adresář s dočasně nahranými moduly, aby mělo prostředí .NET informaci o dalším adresáři s dynamickými knihovnami.

Při kopírování sestavení do dočasněho adresáře je vhodné ho při každé další kompilaci promazat. To se provede automaticky nastavením v Properties projektu modulu v záložce Build Events. Příkaz `rd /s /q "(TargetDir)/../temp/(ProjectName)"` provede jeho smazání. Cesta je samozřejmě závislá na umístění projektu a adresáře pro moduly.

Jenže tím není vyhráno, protože se automaticky nenačtou do aplikace modulární služby, controllery a další implementace. K tomu se využívá IoC Kontejner `Castle.Windsor`, který třídy inicializuje, když je hledá ve hlavní aplikaci *framework MVC*. Pomocí vlastní továrny je pak frameworku poskytne. Implementace používající kontejner je v souboru `Modules.cs`. Zdrojem pro načítání controllerů pomocí IoC Kontejneru byl článek od Mika Kolari [10]. Staví na něm všechna rozšíření vlastního frameworku načítaná pomocí *Dependency Injection*.

Statická třída `Modules` v souboru `Modules.cs` z balíčku `Framework` obsahuje metody, které jsou volány hlavně ve spouštěcí metodě `Application_Start`. Všechny metody využívají společný `Windsor IoC Kontejner`, systematicky prohledávají všechna sestavení modulů a ukládají si implementace rozhraní a specifikace tříd, které jsou poté do aplikace *injektovány*. Metody zároveň hledají Routy pro aplikaci a adaptéry pro validační atributy.

5.4.5 Body rozšíření

V následujících podsekcích jsou probrány podrobně, ale s ohledem na omezení rozsahu, všechny body využité pro rozšíření aplikace pomocí modulů. Některé body jsou mají implicitní implementace již v jádru, aby byla zajištěna správná funkčnost i bez modulů. Během vývoje jsem dospěl k myšlence, že všechny části, které jsou v jádru implicitní, by měly být pro přehlednost také umístěny do oddělených modulů, ale to by vyžadovalo složité přepracování, na které nezbyl během vývoje čas.

5.4.5.1 Routy

Rozšíření routování v aplikaci funguje paralelně s pevně daným routováním jádra. Rozšíření spočívá v dědění rozhraní `IRoutes` třídou v modulu – umístěnou v `Routes.cs`. Třída obsahuje předepsanou metodu, ve které jsou údaje o routách stejné jako v jádru aplikace – viz ukázka 5.6. Načítání rout probíhá ve třídě `Modules` prohledáváním modulů a exekucí předepsaných metod ve třídách.

5.4.5.2 Pohledy

Pohledy poskytují místo pro rozšíření pomocí rodičovských tříd, které ale dědí i pohledy v jádru. Pohledy představují v modulech nový problém. Nemohou totiž být vyhledány jako standardní pohledy v původním adresáři `Views` v jádře. Jsou totiž umístěny spolu se sestaveními v adresáři modulu. Řešením je přepsání metod `View` a `PartialView` v kontroleru,

které cesty k pohledům vracejí. Z tohoto důvodu vznikl controller `PluginController` určený ke specifikaci modulárními controllery. Idea je opět převzata z článku Mika Kolari[10].

Moduly obsahují shodnou strukturu pohledů jako jádro a musí také obsahovat soubor `Web.config` s určením, kterou třídu specifikují a jaké *namespaces* implicitně používají. Jde použít i soubor `_ViewStart.cshtml` v adresáři `Shared` pro všechny pohledy. Soubor může měnit například základní šablonu *layoutu*.

Také jsem přemýšlel o variantě kompilovat pohledy přímo do sestavení, postup byl ale nutný naučit každého, kdo by vyvíjel nový modul. Šlo o process popsany na blogu Chrise van de Steega¹⁴. Pro opuštění této způsobu nahrává i výhoda možnosti úpravy pohledů v nezkompilovaných *cshtml* souborech přímo na serveru.

5.4.5.3 Formuláře

Formuláře jsou obsaženy v pohledech, ale jejich vykreslení s modelem a vázání na model při příjmu je komplikované, proto jsem se zaměřil i na správnou implementaci formulářů v modulech. Mika Kolari[10] úspěšně zkouší ve svém článku použít v modulech formulář. Jeho implementace se nachází v modulu `PageRating` upraveného tak, aby fungoval i v aplikaci e-shopu.

Dále jsem se obával spojení formulářů a widgetů 5.4.5.7. Při testování spolupráce se ale nevyskytl žádný problém. Pro formuláře ve widgetech není třeba žádný speciální postup.

Ještě bych chtěl zmínit už zabudovanou funkčnost do formulářů přímo ve frameworku MVC. Jde o `Display` a `Editor Templates`. Na svém blogu ji popisuje Brad Wilson¹⁵. Bohužel je návod ještě pro verzi MVC 2 a neobsahuje Razor šablony. Myšlenka spočívá v tom, že každá entita může mít svojí vlastní zobrazovací a formulářovou šablonu. Není tedy nutné vypisovat každý prvek entity do pohledu zvlášť. Ukázka 5.10 je z pohledu správy nastavení (šablona `Setting.cshtml`). Kód je ořezaný o HTML tagy tabulky. V souboru `Index.cshtml` je pak šablona použita na řádce 14 pomocí příkazu `Html.EditorForModel`. Framework rozpozná o jakou jde entitu podle deklarace modelu.

5.4.5.4 Controllery

Controllery jsou v modulech samozřejmostí. Jsou rozšiřitelné pomocí třídy `PluginController` zmíněné v části pohledů 5.4.5.2. Bez dědění této třídy nebudou správně v modulech fungovat kvůli zmíněným pohledům.

Controllery jsou do aplikace načítány opět přes třídu `Modules` a využívá se *Dependency Injection*. Načítají se pouze controllery dědící od `PluginController`. Může nastat situace, že některý modul bude obsahovat controller se stejným názvem. Konfliktu se lze vyhnout použitím *namespaces* v odkazech (generovaných příkazem `Html.ActionLink`) i v routách.

Stručná ukázka třídy `PluginController` 5.11 zobrazuje integraci připojení databáze a metod pro lokaci pohledů v modulech. Databázové spojení by mohlo být předěláno na šablonu `Repository` pro Unit testy. Týká se to všech jeho užití. V tom případě by se `Repository` *injektovalo* ve třídě `Modules`. Co se týče hledání pohledů v modulech, využívá se reflexe a

¹⁴<http://www.chrisvandesteeg.nl/2010/11/22/embedding-pre-compiled-razor-views-in-your-dll/>

¹⁵<http://bitly.com/mvc2templates>

```

1 <!--Views/Settings/EditorTemplates/Setting.cshtml-->
2 @model Meshop.Framework.Model.Setting
3 @Html.DisplayFor(item => item.Key)
4 @Html.HiddenFor(item => item.Key)
5 @Html.EditorFor(item => item.Value)
6 @Html.ValidationMessageFor(item => item.Value)
7
8 <!--Views/Settings/Index.cshtml-->
9 @model IList<Meshop.Framework.Model.Setting>
10 @using (Html.BeginForm()) {
11     @Html.ValidationSummary(true)
12     <fieldset>
13         <legend>Settings</legend>
14         @Html.EditorForModel()
15         <input type="submit" value="Save" />
16         @CommonService.Message
17     </fieldset>
18 }

```

Zdrojový kód 5.10: šablony pro formuláře

routovacího slovníku pro nalezení názvů adresářů a souboru. Výjimečné stavy jsou z ukázky pro přehlednost vypuštěny.

```

1 public abstract class PluginController : Controller
2 {
3     protected DatabaseConnection2 db = new DatabaseConnection2();
4
5     public new ViewResult View()
6     {
7         return base.View(ConstructViewPath());
8     }
9     private string ConstructViewPath(string viewName = "")
10    {
11        string viewPath = Modules.Path + GetAssemblyName() + "/";
12        string ctrler = (string)RouteData.Values["Controller"];
13        if (viewName == "") viewName = (string)RouteData.Values["Action"];
14        return viewPath += "Views/" + ctrler + "/" + viewName + ".cshtml";
15    }
16 }

```

Zdrojový kód 5.11: třída PluginController

5.4.5.5 Služby

Služby jsou určeny pro distribuci dat controllerům a pohledům. Definice jejich rozhraní jsou v sestavení Framework. Služby jsou rozděleny podle zaměření:

- **IProduct** – služby týkající se produktu. Jde o vyhledávání produktu v seznamu a další

funkce. Služby v tomto případě představují příklad modelu.

- **ICart** – služba obsahuje všechny příkazy pro manipulaci s nákupním košíkem.
- **ICategory** – služba pro práci s kategoriemi.
- **ICommon** – služba poskytující funkce pohledům jak pro backend tak pro frontend. Jde o hodnoty z nastavení aplikace a informační zprávy. Zobrazení zprávy je vidět v ukázce 5.10 na řádce 16. Služba je injektována do pohledů až za běhu aplikace při kompilaci pohledu s rodičem **BaseWebViewPage**.
- **IFront** – služby pro stránky ve frontendu. Jedná se hlavně o výpis položek hlavního Menu.
- **IAdmin** – služby pro stránky ve backendu. I zde se vypisují položky hlavního Menu.
- **ICheckout** – služba umožňující výměnu procesu *checkoutu* za jiný. V její implementaci lze změnit odkazy na controller zajišťující *checkout*.
- **IListing** – služby pro seznamy zboží vypsanych z kategorií. Tato služba nemá v základní aplikaci využití, protože zobrazování zboží nebylo nijak rozšiřováno.

Předpisy rozhraní by bylo samozřejmě nutné rozšířit v případě nasazení aplikace v ostrém prostředí. Služby jsou opět do controllerů *injektovány* v třídě **Modules**. Pokud není implementace nalezena v modulech, použije se implicitní implementace v sestavení **Core**. Vzhledem k funkcím, které služby mají je možné s nimi nahradit modely.

5.4.5.6 Menu

V aplikaci jsou dvě rozšiřitelná menu. Položky menu jsou odkazy na metody controllerů, které jsou označeny atributem **Menu**. Jejich hledání probíhá v seznamu *controllerů* poskytnutým třídou **Modules**. Pro frontend a backend se načítají menu zvlášť. Implementace jsou ve službách **DefaultFront** a **DefaultAdmin** implementujících rozhraní **IFront** a **IAdmin**. Atribut má možnost pojmenování. Samozřejmostí je překlad názvu podle lokalizace aplikace. Ukázka užití atributu 5.12 popisuje metodu v **HomeController**.

```
1 [Menu(Name = "About")]
2 public ActionResult About()
3 {
4     return View();
5 }
```

Zdrojový kód 5.12: atribut Menu

Position	Template
Left	Index
Right	Product
Bottom	Catalog
Top	Cart
Header	Login
Footer	Address
Document	Shipping
•	Payment
•	Search
•	All

Tabulka 5.1: Možnosti umístění Widgetů

5.4.5.7 Widgets

Widgets jsou grafické prvky v layoutu webové stránky. Dají se odebírat, přidávat a přemísťovat na různá místa různých stránek. Tabulka 5.1 ukazuje možnosti umístění ve stránce i ve specifických pohledech.

Widgets fungují stejně jako běžné metody v controllerech, ale jsou opatřeny tagy `ChildActionOnly` a `Placement`. Potřebují mít tedy i vlastní pohled. Widgets zároveň mohou být v jádru i v modulech.

- `ChildActionOnly` – vestavěný atribut, který dovoluje spustit metodu jako vnořenou, nejde volat přímo pomocí URL.
- `Placement` – určuje na jakém místě se obsah metody objeví. Tabulka 5.1 ukazuje možnosti umístění pomocí parametrů *Position* a *Template* (pohled). Možnosti jsou zapsány v typech enum – `PagePosition` a `PageTemplate`.

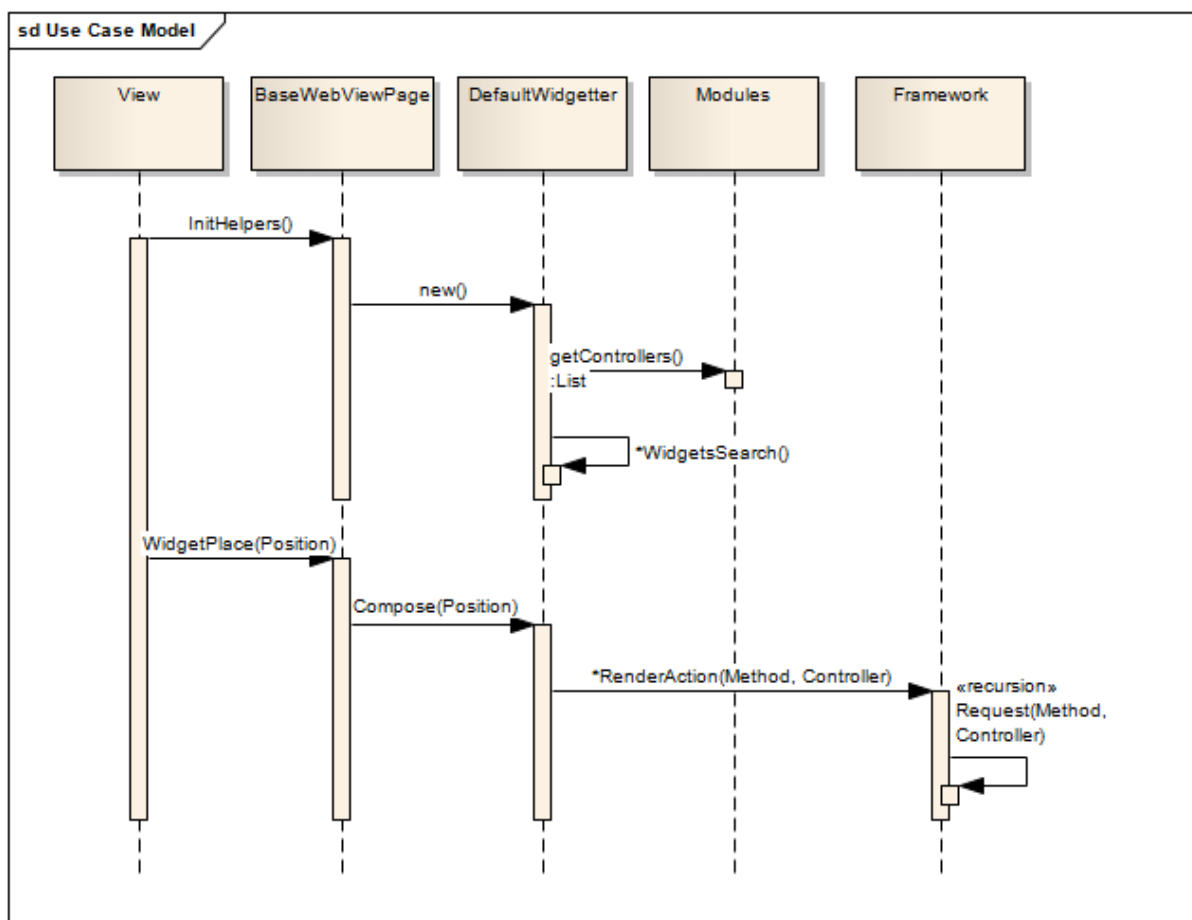
Widgets se načítají z controllerů podobně jako položky menu. Třída `DefaultWidgetter` se načte při načítání stránky, prohledá seznam controllerů a zjistí atributy jejich metod. Poté se na určitých místech ve stránce volají pomocí metody `WidgetPlace`. Ta je ovšem pouze delegátem a implementací je metoda `Compose` ve třídě `DefaultWidgetter`.

V metodě `Compose` se poté volá vestavěná metoda `RenderAction`, která zajistí zobrazení widgetu. Aplikace takto ale prochází znova životním cyklem. Ten se provede tolikrát, kolik je na stránce widgetů. Může se tak jednat o zásadní zpomalení aplikace. Toto je ale jediný dostupný způsob, který poskytuje všechny možnosti akce controlleru i s modelem a formuláři.

Zlepšením implementace by určitě bylo kešování seznamu widgetů a jiný způsob načítání. Pro ilustraci komplexnosti mechanismu je uveden sekvenční diagram 5.6. Stereotyp *«recursion»* představuje nové volání metody v controlleru, tedy nový životní cyklus.

5.4.5.8 Entity

I entity se dají rozšířit, v základu aplikace nebylo této schopnosti využito, ale pro praktickou implementaci je toto třeba. Způsob rozšíření entit je rozdělen interně na 3 způsoby, ale vždy



Obrázek 5.6: Sekvenční diagram zobrazování widgetů

využívá mechanismu dědění původní entity. Detaily nastavení jsou probírány v článku na blogu Morteza Manavi¹⁶.

Připojení demonstruje entita `Review` z modulu `Reviews`. Implementovat injekci entit bylo možné jen díky článku publikovaném jako návodu na psaní modulů pro projekt `NopCommerce` [27]. Nikde jinde, ani v dokumentačních materiálech `Entity Frameworku` tato možnost zmíněna není, což považuji za velkou slabost dokumentace `Entity Frameworku`.

Způsob implementace postupuje podle příkladu z článku [27]. Nejprve je třeba vytvořit třídu s entitou specifikující třídu `PluginEntity`, poté konfiguraci entity podle příkladu specifikující `EntityTypeConfiguration`. Oboje se budou injektovat do `DbContext` v metodě `OnModelCreating`. Dále je možné do aplikace nahrát prvotní obsah entity přes implementaci rozhraní `IDbSeed`, které se injektuje do souboru inicializace databáze. Databáze se bohužel stane nekompatibilní, protože se změní model, viz. popis `Entity Frameworku` 4.9.1.2.

Dotazování na tyto modulární entity z `DbContext` pak probíhá jinak, rozdíl demonstruje ukázka 5.13. Rozšiřující entita je získávána z `DbContext` na řádce 5 a z ní pak základní

¹⁶[<http://bit.ly/CodeFirstInheritance>](http://bit.ly/CodeFirstInheritance)

entita na řádku 6.

Nevýhodou je nemožnost pracovat s rozšiřujícími entitami v jiných modulech o jádře ani nemluvě.

```
1 [ChildActionOnly]
2 [Placement(Position = PagePosition.Document, Template = PageTemplate.Index)]
3 public ActionResult Widget()
4 {
5     var review = db.Set<Review>().First();
6     ViewBag.CustomerMail = review.Customer.Email;
7     return PartialView(review);
8 }
```

Zdrojový kód 5.13: rozdíl použití základní a rozšiřující entity

5.4.5.9 Frontend

Frontend je rozšiřitelný pomocí výše zmiňovaných bodů. Základem rozšiřování je upravování pohledů frontendu, které nejsou zkompilevané. Další možnost je dopisování controllerů a jejich pohledů. Odkazy na akce se přidávají následně do menu frontendu

5.4.5.10 Backend – administrace

Backend, nebo-li administrace, je rozšiřitelná stejně jako frontend s tím rozdílem, že musí být zabezpečená atributem `Admin`. Nemůže totiž dědit současně od `PluginController` a `AdminController`. `PluginController` má proto přednost. Nabízí se možnost vytvoření `AdminPluginController`, který by dědil od jedné třídy a obsahoval manuálně dopsané metody a *properties* druhé třídy.

Controller se poté opět musí přidat do menu backendu stejným atributem `Menu`, rozdíl spočívá v detekci atributu `Admin`.

5.4.6 Omezení modulů

Omezení byla postupně zmiňována v podsekcích Modulů 5.4. Zde jsou shrnuty.

- Entity – rozšíření entit je možné pouze v rámci modulu Mimo něj Entita viditelná není. Viz. 5.4.5.8. Pokud by se entita deklarovala jinde, dojde ke konfliktům a duplikaci.
- Widgety – omezené umístěním na určená místa a omezují výkonnost aplikace, viz. 5.4.5.7.
- AdminPluginController – zmíněný výše v 5.4.5.10. Přesněji jde o vícenásobnou dědičnost, která není v .NET možná. Je třeba sáhnout k rozhraní a duplicitní implementaci.
- Služby – rozšíření služeb musí být provedeno ve frameworku aplikace, který je třeba znova zkompilevat. Modulárním způsobem je možné pouze implementovat rozhraní.

- Kopie do modulu – všem souborům, které se v modulech nekompilují je třeba nastavit manuálně kopírování do adresáře se zkompilovaným modulem.

5.5 Spuštění aplikace

Spuštění aplikace rozlišuje nejméně dva způsoby. Jedním je debug mód, který se spouští přímo ve Visual Studiu a druhým je mód release, který je určený pro ostrý provoz na serveru.

Pro spuštění ve Visual Studiu je třeba následovat některý z instalačních tutoriálů pro ASP.NET MVC 3. Nejlepší je přímo na oficiální stránce¹⁷. Nejlepší volbou je stažení *Web Platform Installer* (WPI) z odkazu v návodu. Ten nabídne instalaci všech základních částí včetně Visual Studia ve verzi zdarma. Poté při otevření **solution** (balík projektů celé aplikace pro Visual Studio) se vyřeší závislosti na jiných knihovnách potřebných k běhu aplikace.

Pro urychlení instalací doplňků je vhodné zaškrtnout při instalaci WPI i instalaci serveru IIS Express a databázového engineu SQL Server Compact.

Pro spuštění na serveru IIS nebo na hostingu je třeba, aby byl server vybaven stejnými technologiemi, tedy ASP.NET 4, MVC 3 a SQL Server Compact. V případě nepřítomnosti frameworku MVC 3 je možné jej nahrát přímo s aplikací. Postup pro nahrání aplikace na IIS je dostupný také v tutoriálech na oficiálních stránkách¹⁸.

Také je důležité, aby server poskytoval plná práva **Full Trust**¹⁹ z důvodu používání IoC kontejneru Castle Windsor, který je vyžaduje. Požadavky na plná práva bohužel vyřazují velké množství webhostingů. Dále je nutné povolit zápis do adresáře `App_Data`, kde je uložen databázový soubor a také zápis do adresáře modulů `extensions/temp`.

5.5.1 Instalace a odebrání modulu

Při instalaci nebo odebírání modulů je třeba vyprázdnit **Application Pool** (keš) v IIS příkazem `Recycle` a restartovat aplikaci. Případně pomůže také přehrát DLL soubory.

Instalace modulu se provádí zkompilováním modulu a spolu se soubory pohledů a dalšími se nakopíruje do aplikace do adresáře `extensions/plugins/Meshop.{Modul}`. Knihovna modulu v tomto případě musí mít název `Meshop.{Modul}.dll`. Zkompile se tak v případě názvu projektu modulu `Meshop.{Modul}`. Při spuštění se adresáře pluginů zkopírují do adresáře `extensions/temp` ze kterých se načtou.

Při odebírání modulu je nutné kromě vymazání keše a restartu aplikace také vymazání adresáře pluginu z obou adresářů `extensions/temp` i `extensions/plugins`.

Data v databázi se při změně v modulech vymažou a načtou se data z třídy `Database-Initializer` a tříd implementujících rozhraní `IDbSeed`. Toto chování je bohužel nastaveno z důvodu absence funkce `Migrations` v Entity Frameworku 4.1, viz. 4.9.1.2.

¹⁷ <<http://asp.net/mvc/mvc3>>

¹⁸ <<http://www.asp.net/mvc/overview/deployment>>

¹⁹ <<http://msdn.microsoft.com/en-us/library/wyts434y.aspx>>

Kapitola 6

Testování

Výkon Render Partial

UX test - Honza, Hozna

Castle neběží na Medium Trust!

- Způsob, průběh a výsledky testování.
- Srovnání s existujícími řešeními, pokud jsou známy.

Kapitola 7

Závěr

7.1 Zhodnocení

Zhodnocení splnění cílů práce a vlastního přínosu práce (při formulaci je třeba vzít v potaz zadání práce).

Nedostatek - roztroušená dokumentace po internetu v návodech, MSDN je poskládaná z fragmentů, knihy pomohly částečně, nezachází tak do hloubky.

7.2 Rozšíření

Diskuse dalšího možného pokračování práce.

dědění modulů, katalogizace.

Literatura

- [1] APEK. *Rekordní Vánoce na internetu* [online]. 2011. [cit. 4.1.2011]. Dostupné z: <<http://www.apek.cz/8477/2176/clanek/rekordni-vanocce-na-internetu/>>.
- [2] BUSCHMANN, F. et al. *Pattern-Oriented Software Architecture, Volume 1 - A System of Patterns*. John Wiley & Sons, 1996. In English.
- [3] Bustamante, M. L. *Extending the ASP.NET 2.0 Resource-Provider Model* [online]. 2006. [cit. 5.21.2011]. Dostupné z: <<http://msdn.microsoft.com/en-us/library/aa905797.aspx>>.
- [4] DEMINICK, S. *Developing a plugin framework in ASP.NET MVC with medium trust* [online]. 2011. [cit. 17.5.2011]. Dostupné z: <<http://shazwazza.com/post/Developing-a-plugin-framework-in-ASPNET-with-medium-trust.aspx>>.
- [5] FOWLER, M. – et. al. *Patterns of Enterprise Application Architecture*. Addison Wesley, 1st edition, 2002. In English.
- [6] Galloway, J. et al. *Professional ASP.NET MVC 3*. WROX, 1st edition, 2011. In English. ISBN 1118076583.
- [7] Gamma, E. et al. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison Wesley, 1998. In English.
- [8] HANSELMAN, S. – GALLOWAY, J. *MVC Music Store* [online]. 2011. [cit. 12.21.2011]. Dostupné z: <<http://www.asp.net/mvc/tutorials/mvc-music-store/>>.
- [9] IEEE Std 1471-2000. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. 2000.
- [10] KOLARI, M. *ASP.NET MVC 3 plugin architecture with embedded Razor views* [online]. 2011. [cit. 20.5.2011]. Dostupné z: <<http://mikakolari.fi/2011/02/aspnetmvc-3-plugin-architecture-with-embedded-razor-views/>>.
- [11] Martin Fowler. *Inversion of Control Containers and the Dependency Injection pattern* [online]. 2011. [cit. 15.12.2011]. Dostupné z: <<http://martinfowler.com/articles/injection.html>>.
- [12] Microsoft. *ASP.NET MVC Framework Overview* [online]. 2011. [cit. 8.12.2011]. Dostupné z: <[http://msdn.microsoft.com/en-us/library/dd381412\(v=VS.98\).aspx](http://msdn.microsoft.com/en-us/library/dd381412(v=VS.98).aspx)>.

- [13] Microsoft. *ASP.NET Overview* [online]. 2011. [cit. 8.12.2011]. Dostupné z: <<http://msdn.microsoft.com/en-us/library/4w3ex9c2.aspx>>.
- [14] Microsoft. *ADO.NET Entity Framework At-a-Glance* [online]. 2011. [cit. 15.12.2011]. Dostupné z: <<http://msdn.microsoft.com/cs-cz/data/aa937709>>.
- [15] Microsoft. *.NET Framework Conceptual Overview* [online]. 2011. [cit. 8.12.2011]. Dostupné z: <<http://msdn.microsoft.com/library/zw4w595w.aspx>>.
- [16] NOURIE, D. *Java Technologies for Web Applications* [online]. 2006. [cit. 8.12.2011]. Dostupné z: <<http://www.oracle.com/technetwork/articles/javase/webapps-1-138794.html>>.
- [17] PHP Documentation authors, see Credits. *PHP History* [online]. 2011. [cit. 8.12.2011]. Dostupné z: <<http://www.php.net/manual/en/history.php.php>>.
- [18] Příspěvatelé Wikipedie. *Design Pattern* [online]. 2011. [cit. 25.2.2011]. Dostupné z: <[http://en.wikipedia.org/wiki/Design_pattern_\(computer_science\)](http://en.wikipedia.org/wiki/Design_pattern_(computer_science))>.
- [19] Příspěvatelé Wikipedie. *Comparison of integrated development environments* [online]. 2011. [cit. 29.11.2011]. Dostupné z: <http://en.wikipedia.org/wiki/Comparison_of_integrated_development_environments#Java>.
- [20] Příspěvatelé Wikipedie. *Modular Programming* [online]. 2011. [cit. 25.2.2011]. Dostupné z: <http://en.wikipedia.org/wiki/Modular_programming>.
- [21] Příspěvatelé Wikipedie. *Perl* [online]. 2011. [cit. 25.2.2011]. Dostupné z: <<http://en.wikipedia.org/wiki/Perl>>.
- [22] Příspěvatelé Wikipedie. *Python* [online]. 2011. [cit. 13.10.2011]. Dostupné z: <<http://en.wikipedia.org/wiki/Python>>.
- [23] Příspěvatelé Wikipedie. *Ruby on rails* [online]. 2011. [cit. 13.10.2011]. Dostupné z: <http://en.wikipedia.org/wiki/Ruby_on_Rails>.
- [24] Příspěvatelé Wikipedie. *Software framework* [online]. 2011. [cit. 25.2.2011]. Dostupné z: <http://en.wikipedia.org/wiki/Software_framework>.
- [25] Provozovatelé two bits s.r.o. *Jak vybírat hosting* [online]. 2011. [cit. 9.12.2011]. Dostupné z: <<http://www.hostingy.cz/navody-a-tipy-jak-vybirat-webhosting.html>>.
- [26] Sanderson, S. – Freeman, A. *Pro ASP.NET MVC 3 Framework*. Apress, 3rd edition, 2011. In English. ISBN 1430234040.
- [27] Severns, S. *nopCommerce Plugin with Data Access* [online]. 2011. [cit. 9.30.2011]. Dostupné z: <<http://blog.csharpwebdeveloper.com/2011/09/23/nopcommerce-plugin-with-data-access/>>.
- [28] Wikipedia Contributors. *Comparison of Web Application Frameworks* [online]. 2011. [cit. 8.12.2011]. Dostupné z: <http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks>.

Příloha A

Seznam použitých zkratek

API Application Programming Interface

IIS

Příloha B

UML diagramy

Mám je sem ještě jednou dát?

Příloha C

Screenshoty aplikace

Příloha D

Instalační a uživatelská příručka

D.1 Instalace prostředí

D.2 Instalace aplikace

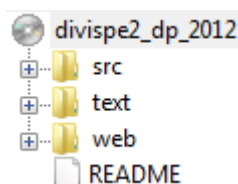
D.3 Spuštění

D.4 Moduly

Příloha E

Obsah přiloženého DVD

Na přiloženém DVD je následující adresářová struktura [E.1](#).



Obrázek E.1: Struktura přiloženého DVD

Soubor **README** je textový a obsahuje tento text, informace jak aplikaci instalovat, spouštět a jaké požadavky má na hardware.

Adresář **src** obsahuje kompletní soubor projektů (solution). To lze otevřít ve Visual Studiu a aplikaci po instalaci všech závislostí spustit přímo ve Visual Studiu.

Adresář **text** obsahuje soubory s vlastním textem práce v PDF formátu a ve formátu \LaTeX s veškerými potřebnými soubory pro kompilaci \LaTeX do PDF.

Adresář **web** obsahuje zkompilevanou aplikaci připravenou pro nasazení na webhosting nebo vlastní IIS server s nainstalovanými závislostmi.

Kompletní obsah DVD je také k dispozici ve webovém repozitáři <http://github.com/czechdude/Meshop>, u kterého není vyloučeno, že se časem rozvine v plnohodnotnou aplikaci.