Vysoká škola ekonomická v Praze
Fakulta informatiky a statistiky
Katedra informačních technologií

Studijní program: Aplikovaná informatika
Obor: Informační systémy a technologie

Přizpůsobení metodiky MMSP pro vývoj webových aplikací v ASP.NET MVC a její využití na reálném projektu

DIPLOMOVÁ PRÁCE

Student : Bc. Filip Velemínský
Vedoucí : doc. Ing. Alena Buchalcevová, Ph.D.
Oponent : Ing. Jan Vít, MBA

2016

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze které jsem čerpal.

V Praze dne 30. listopadu 2016 . . . . . .

Filip Velemínský

Poděkování

Rád bych poděkoval doc. Ing. Aleně Buchalcevové, Ph.D. za odborné vedení mé diplomové práce a konstruktivní připomínky, které vedly k jejímu zkvalitnění. Dále děkuji všem blízkým, za jejich podporu v průběhu celého mého studia.

Abstrakt

Předkládaná diplomová práce se zabývá agilním vývojem webových aplikací na platformě ASP.NET. Hlavním cílem práce bylo vytvoření metodiky MMSP - ASP.NET MVC jakožto úpravy metodiky MMSP, přizpůsobené vývoji ve frameworku ASP.NET MVC. Práce zahrnuje především definice nových a úpravu stávajících objektů metodiky. V rámci textu byly postupně popsány čtyři nově vzniklé Role (např. Databázový specialista), které zodpovídají za vývoj aplikace. Bylo definováno či modifikováno několik Úloh (např. Návrh databáze) a Pracovních produktů (např. Datový model), za které nové Role zodpovídají. Všechny přidané objekty byly poté do metodiky integrovány úpravou jejího Životního cyklu.

Práce byla v první části obecně zaměřena na vývoj webových aplikací, definování jejich specifik a analýzu možností využití agilního vývoje. Byla představena metodika MMSP a ověřena její vhodnost pro plánované rozšíření. Dále byl představen samotný framework ASP.NET MVC, jeho architektura a základní charakteristiky. Následuje samotná úprava metodiky, která byla navíc v poslední kapitole ověřena na příkladu vývoje aplikace pro správu databáze antropologického oddělení Národního muzea v Praze.

Klíčová slova

Metodika pro malé softwarové projekty (MMSP), Model View Controller (MVC), ASP.NET MVC 5, webová aplikace.

Abstract

The subject of this thesis was agile development on ASP.NET web development platform. The main goal was to create new methodology MMSP - ASP.NET MVC by adjusting methodology MMSP for development in ASP.NET MVC framework, which was mainly achieved through the definition of new objects in the methodology. Text describes four newly created roles (eg. Database specialist) which are responsible for application development. Text also defines or modifies several tasks (eg. Database design) and work products (eg. Data model). All added objects were integrated into the methodology by adjusting its life cycle.

At first, thesis was generally focused on the development of web applications, defined their characteristics and analyzed the possibilities of using agile development. Then the MMSP methodology was presented and verified its suitability for the planned extension. Furthermore, there was described the ASP.NET MVC framework, its architecture and basic characteristics. In the last chapter of this thesis methodology MMSP-ASP.NET MVC was verified on the example of development database management web application for the Anthropological Department of the National Museum in Prague.

Keywords

Methodology for small software projects (MMSP), Model View Controller (MVC), ASP.NET MVC 5, web application.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

1 Úvod

Vývoj webových aplikací zaznamenal v posledních letech veliký rozvoj, stejně jako většina oblastí informačních technologií. Prvotní statické webové stránky se v průběhu dvou desetiletí radikálně transformovaly v různá řešení, jako jsou elektronické obchody, podnikové aplikace, či celosvětově užívané sociální sítě. Tento vývoj stále pokračuje a je zřejmé, že se spolu s ním mění i používané technologie a jejich technické zázemí. Zvyšují se zároveň i samotné nároky na webové aplikace. Moderní informační technologické procesy by tudíž měly být časově nenáročné, měly by být konstruovány tak, aby je bylo možné snadno udržovat a doplňovat o nové algoritmy podle měnících se a přibývajících požadavků.

Jedním z faktorů, který ovlivňuje splnění výše uvedených nároků na produkt, umožňuje se vyvarovat známých problémů při vývoji a do jisté míry ovlivňuje úspěšnost celého projektu, je zvolená metodika. Tato diplomová práce se zabývá oblastí vývoje webových aplikací, konkrétně způsobem jejich tvorby a údržby v technologii ASP.NET MVC 5 v rámci menších projektů. Předmětem práce je analýza Metodiky pro Malé Softwarové Projekty (dále MMSP), její přizpůsobení pro vývoj v technologii ASP.NET MVC a následné ověření na konkrétním příkladu. Metodika MMSP vznikla lokalizací metodiky OpenUP v rámci diplomové práce (Rejnková, 2011) a je popsána také v rámci knihy *Příklady modelů analýzy a návrhu aplikace v UML* (Buchalcevová a Stanovská, 2013).

Model view controller (dále MVC) je oblíbený architektonický vzor, který ačkoli původně vznikl na desktopech, své uplatnění našel zejména u webových aplikací. Kromě ASP.NET MVC je součástí i jiných webových frameworků, jako je např. Zend a Nette pro PHP, či Ruby On Rails pro Ruby. S technologií ASP.NET MVC jsem se poprvé seznámil v rámci předmětu 4IT449 - Vývoj aplikací na platformě .NET, a protože mě zaujala, dále jsem se jí věnoval také ve volném čase. Jedná se však o poměrně novou technologii, která zaznamenává rychlý vývoj a současné metodiky pro vývoj softwaru by jí mohly být více přizpůsobeny - ať už z pohledu stanovení pravidel pro vývoj, z pohledu týmové spolupráce, či dokumentace. V rámci diplomové práce jsem se tedy rozhodl své znalosti v dané oblasti prohloubit a zároveň přizpůsobit vhodnou dosavadní metodiku tak, aby usnadnila a zkvalitnila vývoj webových aplikací v této technologii.

1.1 Cíl práce

Hlavním cílem diplomové práce bylo upravit metodiku MMSP pro vývoj webových aplikací v ASP.NET MVC. V práci je definován jednotný přístup k vývoji těchto aplikací, který přispěje ke zkvalitnění celého procesu a k eliminaci častých problémů.

Pro splnění hlavního cíle práce jsem definoval dílčí cíle:

- charakterizovat specifika vývoje webových aplikací,
- charakterizovat agilní přistup k vývoji webu a metodiku MMSP,
- analyzovat možnosti aplikace metodiky MMSP při vývoji webových aplikací,
- charakterizovat framework ASP.NET MVC 5 a jeho možnosti v oblasti vývoje webových aplikací,
- upravenou metodiku publikovat pomocí nástroje Eclipse Process Framework Composer (EPFC),
- ověřit upravenou metodiku na konkrétním případě - aplikace pro správu databáze antropologického oddělení Národního muzea (AONM).

1.2 Cílová skupina

Cílovou skupinou této práce jsou osoby zabývající se vývojem webových aplikací v technologii ASP.NET MVC, a to jak v rámci školních či reálných projektů, tak na teoretické úrovni.

Podobně jako metodika MMSP, také tato práce je určena především pro členy menších projektových týmů, zastávajících roli vývojáře, analytika, projektového manažera či testera. Práce může poskytnout cenné informace i dalším zainteresovaným stranám mimo projektový tým. Např. novým zákazníkům je tak možné podat představu, jakým procesem vývoje bude webová aplikace procházet a pomoci jim lépe pochopit i svou roli v tomto procesu.

1.3 Použité metody

Nejprve bylo nutné určit vhodnost využití agilního přístupu pro vývoj webových aplikací pomocí frameworku ASP.NET MVC. Toho bylo dosaženo metodou komparace klasického, rigorózního přístupu právě s agilním.

Následuje analýza metodiky MMSP. Cílem analýzy bylo zjistit silné stránky metodiky, na kterých je možné úpravu stavět a naopak upozornit na části metodiky, které se pro vývoj webových aplikací moc nehodí, které je třeba přepracovat či doplnit. Současně byl analyzován i samotný framework ASP.NET MVC s cílem zjistit, jaké postupy, pravidla a nástroje je vhodné do nově vznikající metodiky zahrnout. Jednotlivé principy byly vybírány mimo jiné i v návaznosti na prostudování jiných metodik a jejich náležitostí, či dle osobních zkušeností s vývojem.

Na základě předchozí analýzy byla pomocí syntézy prováděna samotná úprava metodiky – přidání Rolí, Činností a procesů Řízení. Kromě specifikace nových prvků metodiky byly představeny i modely (příklady) ideálních Pracovních produktů. Upravená metodika byla následně ověřena metodou pilotního projektu, vývojem webové aplikace. Protože v průběhu vývoje aplikace nejspíše vyvstanou nové potřeby úpravy metodiky a naopak, budou tyto činnosti prováděny v několika iteracích. Tím bude dosaženo konzistence a vyšší míry adaptability nové metodiky.

1.4 Struktura práce

První kapitolou je úvod diplomové práce, jehož účelem je představení obsahu a hlavních cílů práce.

Druhou kapitolou je komentovaná rešerše, ve které jsou shrnuty hlavní zdroje, které byly v průběhu psaní diplomové práce využity. Cílem kapitoly je nalézt a analyzovat kvalitní, důvěryhodné informační zdroje.

Třetí kapitola se nejprve zabývá obecně agilním vývojem. Následně jsou popsána specifika vývoje webových aplikací a možnosti využití právě agilního přístupu. V poslední části kapitoly je představena metodika MMSP a jsou specifikovány důvody jejího výběru pro následné úpravy.

Ve čtvrté kapitole je představen framework ASP.NET MVC. Cílem této kapitoly je charakterizovat zmíněný framework pro vývoj webových aplikací a zjistit, jaké postupy, pravidla a nástroje je vhodné do nově vznikající metodiky zahrnout. Kapitola popisuje např. využitou architekturu, historii frameworku a jeho hlavní výhody.

Pátá kapitola se zabývá návrhem a úpravou metodiky MMSP – ASP.NET MVC. Pro přehlednost je návrh rozložen do pěti podkapitol, odpovídajících jednotlivým prvkům metodiky definovaných v třetí kapitole. V poslední podkapitole je popsán proces publikace metodiky MMSP – ASP.NET MVC pomocí nástroje EPFC.

V šesté kapitole je popsán proces ověření metodiky MMSP – ASP.NET MVC na příkladu vývoje webové aplikace pro správu databáze antropologického oddělení Národního muzea v Praze. Nejprve je specifikován účel aplikace a její budoucí využití. Následně jsou představeny jednotlivé Pracovní produkty, vniklé na základě provádění Úloh metodiky definovaných v rámci páté kapitoly, spolu s popisem jejich vzniku.

Poslední, sedmou kapitolou je závěr, ve kterém jsou zhodnoceny stanovené cíle. Je zhodnoceno, čeho je v rámci práce dosaženo, jaký je její přínos a využitelnost dosažených výsledků.

1.5 Předpoklady a omezení práce

Práce je zaměřena, stejně jako metodika MMSP, specificky na malé týmy o dvou až deseti lidech, ve kterých nejsou vyvíjeny kritické aplikace.

Cílovou skupinou čtenářů této práce jsou v podstatě všechny role pracující na vývoji webové aplikace. Největší důraz je však kladen na činnosti vývojáře, které v metodice MMSP chybí, či je bylo nutno přizpůsobit dané technologii.

Vzhledem k povaze práce je pravděpodobné, že bude obsah ovlivněn subjektivními názory, vycházejícími ze současných zkušeností s vývojem aplikací v dané technologii. Jedná se např. o přístup k řešení jednotlivých kroků vývoje či neúplnost znalostí některých oblastí.

Při samotné úpravě stávající metodiky je nutné zachovat její konzistenci a nenarušit posloupnost jednotlivých procesů.

1.6 Přínos práce

Hlavním výstupem diplomové práce je upravená metodika MMSP pro vývoj webových aplikací s využitím frameworku ASP.NET MVC, kterým se dosud žádná metodika nezabývala. Upravená metodika nazvaná MMSP-ASP.NET MVC má využití především u méně zkušených týmů vyvíjejících ve zmíněném frameworku, lze ji však také z velké části využít při vývoji webových aplikací v jiné technologii.

Další přínos práce spočívá v implementaci aplikace pro správu databáze antropologického oddělení Národního muzea. Zde jsem navázal na svou bakalářskou práci (Velemínský, 2014), kde jsem analyzoval požadavky antropologického oddělení a zmíněnou databázi navrhl. I v případě vývoje této aplikace jsem úzce spolupracoval se zaměstnanci AONM. Aplikace je tak vytvořena dle jejich požadavků na funkcionalitu i ovládání a bude dále reálně využívána jako hlavní uložiště informací o jejich sbírkách.

2 Komentovaná rešerše informačních zdrojů

V této kapitole jsou shrnuty hlavní zdroje, které byly v průběhu psaní diplomové práce využity. Cílem kapitoly je nalézt a analyzovat kvalitní, důvěryhodné informační zdroje. Jedná se o zdroje zaměřující se především na technologii ASP.NET MVC a o zdroje týkající se rozšiřované metodiky MMSP, případně metodik pro vývoj softwaru obecně.

2.1 Odborná tištěná literatura

Odborná tištěná literatura je hlavním zdrojem diplomové práce.

Důležitým zdrojem z oblasti webových technologií je kniha *Professional ASP.NET MVC 5* (Galloway et al., 2014), která se zaměřuje na funkcionalitu poslední stabilní verze MVC 5. Hlavní autor výše zmíněné publikace pracuje jako technický evangelista Microsoftu zaměřující se přímo na ASP.NET. Tím se dostává do kontaktu jak s profesionály, tak se začátečníky této technologie a v knize tyto zkušenosti uplatňuje. V knize je vysvětleno mnoho technik vývoje, které mají za cíl představit a uplatnit největší výhody této technologie. Součástí je mnoho návodů a praktických ukázek pokrývajících téměř veškerou funkcionalitu frameworku. Kniha je určena jak začátečníkům, tak pokročilým uživatelům a v kontextu diplomové práce je využita jako základy dané technologie, na které bylo v dané práci navázáno.

Další zdroj, který se zaměřuje na vývoj ve zvoleném frameworku, je kniha *Pro ASP.NET MVC 5* (Freeman, 2013). Autor knihy je zkušený IT profesionál, který pracoval na mnoha seniorských pozicích, např. jako Technický ředitel (CTO) světové banky. Kniha, stejně jako předchozí, počítá se základními znalostmi jazyka C#, HTML a CSS. V první části knihy je Framework představen, jsou vysvětleny jeho přednosti a zařazení v současném trendu webového vývoje. Následně se celá kniha věnuje jednomu projektu – sportovnímu internetovému obchodu, jehož vývoj je představen od fáze zahájení po fázi zavedení a jsou při něm představeny všechny hlavní funkčnosti, které Framework nabízí. Při psaní práce bylo využito především druhé části knihy, ve které jsou jednotlivé funkčnosti a mechanismy MVC Frameworku, použité při vývoji zmíněného obchodu, vysvětleny z pohledu jak fungují a jak je možné je konfigurovat a přizpůsobit.

Jako jeden z hlavních zdrojů z oblasti metodik vývoje softwaru je využita kniha *Příklady modelů analýzy a návrhu aplikace v UML* (Buchalcevová a Stanovská, 2013). Kniha podrobně popisuje metodiku MMSP, která bude v rámci diplomové práce upravena. Za výhodu knihy považuji nejen přehlednou charakteristiku celé metodiky, rozdělenou do logicky oddělených celků, ale také využití samotné metodiky na dvou odlišných

projektech. Díky tomu je snadnější metodiku uchopit i v případě absence praktických zkušeností.

2.2 Elektronické informační zdroje

Dalšími zdroji informací použitými v této práci jsou zdroje internetové. Jedná se buď o specializované portály, komunitní fóra či elektronické články. K jejich vyhledání využívám především vyhledávač *Google* (Google, 2016) a portál pro vyhledávání zdrojů Vysoké školy ekonomické v Praze (VŠE, 2016).

Jedním z použitých elektronických zdrojů je oficiální dokumentace k frameworku ASP.NET MVC od společnosti Microsoft (ASP.NET MVC, 2016). Využívána byla k doplnění, či ujasnění znalostí o daném Frameworku a také v průběhu implementace aplikace. Její přinos spočíval především v mnoha praktických příkladech základních řešení, která obsahuje téměř každá aplikace.

Dalším zajímavým portálem o webových technologiích je *TechFunda.com* (TechFunda™, 2016). Obsahuje velké množství krátkých ukázek kódu, rozdělených přehledně dle patřičných kategorií. Portál je v průběhu psaní práce využit nejen pro získání informací o ASP.NET MVC, ale také o HTML5, JavaScriptu či jQuery, což jsou technologie které je nutné při vývoji webu alespoň zběžně znát a umět použít.

Z rešerše českých webových stránek zmíním portál *ITNetwork* (ITnetwork.cz, 2015), který se věnuje mnoha technologiím, především v oblasti vývoje webu. Samozřejmě se tak věnuje i frameworku ASP.NET MVC. Je zde možné nalézt mnoho tutoriálu pro začátečníky i pokročilé, včetně zdrojových kódů. Nevýhodou je fakt, že více specializované články jsou na tomto portále zpoplatněny. To je však kompenzováno jejich kvalitou.

Za velice přínosný zdroj, využitý v rámci samotné úpravy metodiky MMSP, považuji metodiku *RUP for Small Projects* (RUP, 2006) a metodiku *OpenUP* (OpenUP, 2012). Jedná se o osvědčené metodiky vývoje softwaru, přičemž samotná metodika MMSP vznikla lokalizací a úpravou metodiky OpenUP. Obě metodiky byly použity při analýze možností rozšíření MMSP a to nejen z hlediska obsahu, ale i struktury. Pomocí nich tak jsou v upravené metodice zachovány všeobecné konvence, ať už se jedná o pojmenování jednotlivých kategorií či jejich návaznost a zpracování.

2.3 Akademické práce

Stěžejní zdroj, ze kterého jsem při úpravě metodiky MMSP vycházel, je diplomová práce *Lokalizace a přizpůsobení metodiky OpenUP* (Rejnková, 2011). V práci je popsána metodika MMSP, kterou autorka práce vytvořila lokalizací metodiky OpenUP a její částečnou úpravou. Kromě samotné metodiky autorka popisuje i nástroj EPF Composer, který slouží ke správě metodik a který jsem v rámci své práce také využil.

Dalšími zdroji, kterými je tato práce ovlivněna, jsou diplomové práce zabývající se úpravou metodiky MMSP (Kábrt, 2014; Novotný, 2015; Nývlt, 2012). Hlavní přínos těchto prací pro mě spočívá v technice, jak úpravu metodiky zpracovat a v některých případech i využití upravených částí metodik, na které dále v textu odkazuji.

Jedná se např. o práci Rozšíření metodiky MMSP v oblasti analýzy a návrhu testování (Novotný, 2015). Autor metodiku rozšiřuje o značný počet analytických činností v oblasti testování, přidává nové role a popisuje doporučené techniky testování.

Další prací zabývající se úpravou metodiky MMSP je Využití metodiky MMSP při vývoji IS v prostředí FileMaker (Kábrt, 2014). V tomto případě práce pojednává o úpravě metodiky pro vývoj v konkrétním softwaru. Ačkoli se jedná naprosto jinou technologii, má pro mě práce přínos ve formě pojetí úpravy a v možnostech rozšíření metodiky pro konkrétní vývojový nástroj.

Zajímavá je také diplomová práce Metodika pro vývoj webových aplikací (Mittner, 2011). Autor v ní analyzuje možnost použitelnosti agilních metodik pro vývoj webové aplikace v PHP. Následně upravuje jím zvolenou nejvhodnější metodiku (OpenUP), pro lepší využitelnost v rámci vývoje webových aplikací. V rámci práce jsou definovány nové role, produkty, činnosti a různé návody či nástroje pro usnadnění vývoje v PHP. Ačkoli se jedná o jinou technologii, některé principy zůstávají při vývoji webu stejné a byly tak použity i v rámci mojí úpravy metodiky.

Prací zaměřujících se na samotný Framework ASP.NET MVC není mnoho, většinou se navíc jedná o starší verze tohoto frameworku. Poměrně zajímavá je práce *Webová aplikace založená na frameworku ASP.NET MVC 3/RAZOR* (Kvapil, 2012). Autor se zde téměř nezabývá teoretickou částí, zato důkladně popisuje vývoj jeho aplikace, včetně všech použitých komponent. Je zde tak například popsána práce s technologiemi jQuery či AJAX a kromě přístupů zvolených autorem jsou vždy zmíněny i jiné možnosti řešení.

Poslední diplomová práce, kterou zde zmíním, nese název *Lokalizace Vstupního profilu normy ISO/IEC 29110 do češtiny a jeho publikace v Eclipse Process Framework Composer* (dále EPFC), (Šimko, 2015). V práci se autor mimo jiné věnuje detailní charakteristice konceptu Unified Method Architecture (UMA) a nástroje pro správu a údržbu metodiky EPFC, který je na zmíněném konceptu postaven a který v rámci práce také využívám. Za

velký přínos práce pokládám přeložení všech komponent a kategorií, se kterými se v nástroji pracuje. Zároveň jsou přínosné praktické zkušenosti s použitím aplikace – autor zde popisuje řešení nedostatků EPFC, jako jsou např. chybějící metodické prvky či možnost vyhnout se anglickým názvům ve struktuře projektu.

3 Agilní přístup k vývoji webových aplikací

Společně s vývojem softwaru se i webový vývoj posunul směrem k agilním metodikám. Toto tvrzení se může vykládat různými způsoby, nicméně jde hlavně o řízení softwarových projektů jako adaptivních procesů a nepodléhat nadměrnému plánování. Agilní metodiky jdou ruku v ruce se souborem vývojových postupů a nástrojů, které tyto praktiky podporují (Freeman, 2013).

V této kapitole nejprve krátce popíši principy agilního vývoje a důvody, proč je jejich využití při vývoji webových aplikací vhodné. Následně uvedu specifika vývoje webových aplikací, především čím se odlišují od obecného vývoje software. V poslední části kapitoly představím metodiku MMSP, uvedu její charakteristiku a důvody, proč byla zvolena jako základ pro rozšíření, zpracovávané v této práci.

3.1 Agilní vývoj

Základem agilního vývoje je iterativnost a týmový přístup. Snaha je dosáhnout co nejrychlejšího dodání aplikace implementované po menších celcích. Všechny agilní techniky se opírají o priority definované v manifestu agilního vývoje (Fowler a Highsmith, 2001), které dávají přednost:

- individualitám a komunikaci před procesy a nástroji,
- provozu schopnému software před obsažnou dokumentací,
- spolupráci se zákazníkem před sjednáváním kontraktu,
- reakci na změnu před plněním plánu.

Základní principy a také myšlenku celého agilního přístupu dodržují všechny známé agilní metodiky, existuje však celá řada jejich interpretací. V knize *Agile!: The Good, the Hype and the Ugly* (Meyer, 2014) jsou principy agilního vývoje přehledně a výstižně shrnuty do celkem osmi bodů, rozdělených do dvou skupin:

organizační

1. Postavit zákazníka do centra dění.
2. Nechat tým sebe-organizovat.
3. V průběhu práce zajistit udržitelné tempo.
4. Vyvíjet minimální software:
    a) implementovat minimální funkcionalitu,
    b) implementovat jen to, co je požadováno,

c) vyvíjet pouze kód a testy.

5. Přijímat změny.

technické

6. Vyvíjet iterativně:
    a) vytvářet časté pracovní iterace,
    b) neměnit požadavky během iterace.
7. Chovat se k testům jako ke klíčovému zdroji:
    a) nezačínat vyvíjet další část, dokud neprojdou všechny testy,
    b) vytvářet nejdříve testy.
8. Vyjadřovat požadavky pomocí scénářů

Existuje mnoho důvodů, proč jsou agilní metodiky pro vývoj webových aplikací vhodnější než klasické, rigorózní metodiky. Některé plynou přímo z principů definovaných výše, jiné však na první pohled jasné být nemusí. Webové aplikace se obvykle vyvíjejí v menších týmech, což je dělá ideálními adepty pro využití agilních praktik. Zákazník obvykle neví, co je možné implementovat a především v počátečních fázích ani nemusí mít ujasněno, co implementovat chce. Agilní vývoj umožní rychle vytvořit jednoduchý základ webových stránek a ten následně iterativně vyvíjet na základě zpětné vazby od reálných uživatelů a jejich potřeb. Staví se pozitivně k změnám, bez kterých je kvalitní, použitelnou webovou aplikaci téměř nemožné implementovat. Díky agilnímu přístupu se samotný produkt začíná dříve využívat a obecně je u agilního vývoje prokázána kratší doba trvání celého projektu než u tradičního přístupu. V neposlední řadě se využitím agilních přístupů limitují rizika a investice.

Další skutečností, hovořící ve prospěch agilního vývoje, je rychle se měnící technologie pro webový vývoj a fakt, že se vše promítá v jednom uživatelském rozhraní – webovém prohlížeči. S tím jsou spojeny i obvyklé velké rozdíly mezi zkušenostmi vývojářů.

Vývoj ve frameworku ASP.NET MVC je možné provádět dle agilní, ale i rigorózní metodiky. Výběr závisí na preferencích a složení implementačního týmu. Tato práce se zabývá projekty pro menší týmy využívajících přístupu agilního.

3.2 Specifika vývoje webových aplikací

Postupem času se rozdíl mezi vývojem webových aplikací a klasickým vývojem softwaru zmenšuje. Na webové aplikace se kladou stále větší nároky a nahrazují mnoho desktopových aplikací. Při jejich vývoji je však nutno vzít v potaz řadu skutečností, kterými se od vývoje ostatního software liší.

3.2.1 Omezení technologiemi

Hlavní odlišností při vývoji webových aplikací je bezstavovost HTTP protokolu, který slouží se komunikaci prohlížeče a serveru. Termínem bezstavový se rozumí, že protokol neumí udržovat žádný kontext a spojit si několik po sobě jdoucích požadavků do logické sekvence.

Při vývoji webových aplikací je obecně doporučeno navrhovat aplikace jako bezstavové, pokud to je jen trochu možné. Nesnažit se tuto vlastnost obejít, ale naopak ji využít. V případě, že je nutné se stavem v aplikaci pracovat, je k dispozici několik variant implementace: Cookies, Sessions či ViewState. V případě Cookies se ukládá malé množství dat u klienta a při požadavku se přenášejí na server. Sessions fungují tak, že přenáší identifikátor dané session a vlastní data jsou uložená na serveru. V případě ViewState jsou potřebná data uložená v zašifrované podobě ve skrytém poli formuláře (MSDN, 2016).

Další úskalí tkví v neustálém vývoji webových technologií – ať už protokolů jako DNS, HTTP, či emailových, tak technologií vývoje jako CSS, DHTML, JavaScript, Flash, v neposlední řadě i webových prohlížečů. Webový vývojář by měl být schopen udržovat své znalosti o používaných technologiích aktuální.

3.2.2 Důraz na bezpečnost

V případě vývoje webových aplikací je nutno klást větší důraz na bezpečnost. Webové aplikace jsou náchylnější k potencionálním útokům a vyžadují tak vyšší stupeň bezpečnosti. Tato bezpečnost zahrnuje ošetření URL parametrů, Cookies, formulářů a všeho, co s nimi souvisí.

Nezabezpečená webová aplikace může rychle oslabit bezpečnost celé sítě obejitím všech bezpečnostních opatření. Bezpečností firewall ani jiné ochranné funkce většinou neochrání uživatele před bezpečnostními chybami aplikace. Zabezpečení webové aplikace je komplikovaný proces zahrnující proaktivní a reaktivní opatření (Harwood, 2015).

3.2.3 Různorodost klientských zařízení

Dalším specifikem při vývoji webových aplikací je nutnost využívání takových služeb a technologií, které jsou podporovány všemi hlavními webovými prohlížeči. Prohlížeče se sice stále vyvíjejí a podporují řadu nových standardů, zároveň však musí zachovávat zpětnou kompatibilitu pro starší weby. Vývoj každého prohlížeče se navíc ubírá trochu jiným směrem a webový vývojář musí používat funkčnosti podporované všemi prohlížeči, nebo je pro jednotlivé prohlížeče implementovat zvlášť. V posledních letech se navíc musí u většiny webových aplikací počítat i s přístupem z mobilních zařízení, kde jsou možnosti prohlížečů přeci jen omezenější.

3.2.4 Architektura klient-server

Webový vývoj je založený na komunikaci mezi klientem a serverem. Server je zodpovědný za poskytování stránek, klient naopak vytváří požadavky na tyto stránky a následně je zobrazuje uživateli – ve většině případů je klientem webový prohlížeč. Programování pro každou část vyžaduje jiné technologie a přístupy. Obvykle se při vývoji webových aplikací využívá klient/server architektury s tenkým klientem, kdy jsou téměř všechna data uložena na serveru a provádí se tam i většina logického zpracování. Klient pak není zatěžován a slouží jen jako jakýsi zobrazovač stránek poslaných serverem. Klientskému programování se však v moderních webových aplikacích nelze vyhnout – díky němu jsou stránky interaktivní, dynamické, je možné ukládání dočasných dat, posílat požadavky apod. (TechFunda™, 2014). Webový vývojář tak musí řešit, co bude zpracováno na straně klienta, co na straně serveru a s tím spjatou optimalizaci.

3.2.5 Rychlost

Důležitou stránkou webové aplikace, na kterou je třeba při jejím vývoji dbát, je rychlost. Aplikaci běžící na jenom serveru využívá současně několik uživatelů a jakékoli delší prodlevy uživatele od používání aplikace odradí. Hlavní roli zde tak hraje optimalizace aplikace, přičemž existuje mnoho možností, jak ji provádět – např. minimalizací HTTP požadavků, snížením velikosti externích souborů, optimalizací dotazů do databáze atd.

3.2.6 Rozšiřitelnost

Po osvědčení aplikace v praxi se obvykle začíná pracovat na jejích úpravách, vylepšeních či nadstavbách. Je dobré počítat s těmito modifikacemi již v průběhu návrhu aplikace a řešit většinu složitějších webových aplikací modulárním způsobem.

3.3 Metodika MMSP

Metodika pro malé softwarové projekty byla vytvořena v rámci diplomové práce (Rejnková, 2011), jako lokalizace a přizpůsobení metodiky OpenUP. Dále byla popsána v publikaci *Příklady modelů analýzy a návrhu aplikace v UML* (Buchalcevová a Stanovská, 2013). Metodika je určena především pro projekty probíhající v rámci výuky na VŠE v Praze, ale může být využívána i na reálných projektech z praxe. Metodika je vhodná pro menší týmy o zhruba 10 členech týmu a pro nekritické projekty s dobou trvání do 6 měsíců. Metodika je veřejně dostupná na portálu kitscm.vse.cz.

Metodika MMSP je tvořena množinou prvků, jejichž vzájemné vazby vytvářejí vlastní proces vývoje softwaru. Seznam prvků s jejich definicí obsahuje tabulka níže (MMSP, 2011).

Tabulka 1: Prvky metodiky MMSP (zdroj: MMSP, 2011)

Název	Popis	**P**říklad
Role	Charakteristika člena vývojového týmu, který provádí v rámci procesu vývoje jednotlivé úlohy a vytváří požadované pracovní produkty. Role definují schopnosti a zodpovědnosti, ne konkrétní osoby.	Vývojář, Analytik
Úloha	Popis, jak pracovat, aby bylo dosaženo stanovených cílů či byly vytvořeny určité pracovní produkty. Úlohy jsou vykonávány rolemi a jsou obvykle definovány jako série dílčích kroků.	Naplánování iterace, Vytvoření vize projektu
Proces	Popis struktury práce a workflow činností, který spojuje dohromady úlohy, role a pracovní produkty.	Testování
Pracovní produkt	To, co je v rámci procesu vývoje vytvářeno, modifikováno či používáno.	Databáze, vize, zdrojový kód
Návod	Šablony, příklady, rady a další pomůcky, které je možné v rámci procesu vývoje využít.	Šablona plánu iterace, Vzorový projektový plán

Základem metodiky je životní cyklus projektu, který je rozdělen stejně jako metodika RUP či OpenUp na 4 fáze. Jsou to fáze Zahájení, Rozpracování, Konstrukce a Zavedení. Fáze Zahájení se soustředí na porozumění požadavků, identifikování klíčových funkcí a označení kritických rizik projektu. Ve fázi Rozpracování je kladeno za cíl porozumění požadavkům, vymezení architektury a upřesnění plánu projektu. Samotné vytvoření funkční verze softwarového systému spadá do fáze Konstrukce. V poslední fázi s názvem Zavedení jsou opravovány případné chyby a systém je uveden do provozu

(Buchalcevová a Stanovská, 2013). Důležitý je také fakt, že v rámci kterékoli fáze může být provedeno více iterací, které se mohou lišit jak podrobností prováděných úloh, tak délkou. Největší počet iterací obvykle náleží fázi Konstrukce, kdy se hodí systém vyvíjet po menších částech.

Další oblastí definovanou v MMSP jsou tzv. Disciplíny. Ty představují jakési seskupení Úloh a Pracovních produktů metodiky podle jejich oblasti zájmu. Metodika MMSP definuje následující Disciplíny:

- Řízení projektu,
- Řízení změn a konfigurací,
- Požadavky,
- Architektura,
- Vývoj,
- Testování.

V rámci této práce se budu vzhledem ke stanovenému cíli soustředit především na Disciplínu Vývoj, ve které je popsáno, jakým způsobem by mělo být navrženo a implementováno technické řešení produktu. *„Hlavním smyslem této disciplíny je zajistit, aby veškeré požadavky, jak funkční, tak nefunkční, byly transformovány do vytvářeného softwarového systému“* (Buchalcevová a Stanovská, 2013). Do Disciplíny Vývoj patří Úlohy Návrh řešení, Implementace řešení, Integrace a vytvoření buildu (sestavení), Tvorba a Provedení unit testů.

Metodika MMSP je metodikou vývoje softwaru, kterou v rámci diplomové práce upravuji pro její lepší využitelnost při vývoji aplikací ve frameworku ASP.NET MVC (viz kapitola 5). Důvodů jejího výběru je několik:

- Česká lokalizace metodiky – jedná se o metodiku dostupnou v českém jazyce.
- Doporučení od vedoucího diplomové práce – doporučení, abych jako výchozí metodiku pro následné úpravy využil právě MMSP.
- Dostupnost zdrojů – metodika je podrobně popsána v diplomové práci v rámci které vznikla (Rejnková, 2011) a v knize *Příklady modelů analýzy a návrhu aplikace v UML* (Buchalcevová a Stanovská, 2013).
- Didaktická povaha metodiky – metodiku je možné dále využívat či rozšiřovat v rámci výuky na VŠE v Praze.
- Podpora úprav - metodika je vytvořena jako plug-in k metodice OpenUP a může být dále snadno upravována prostřednictvím nástroje EPFC.

4 Framework ASP.NET MVC

Cílem této kapitoly je charakterizovat framework ASP.NET MVC umožňující tvorbu webových aplikací v jazyce C#. Nejprve je představena samotná architektura MVC, na které je framework postaven. Následně je krátce popsán programovací jazyk C# a charakterizována platforma ASP.NET. Poslední část kapitoly se věnuje popisu samotného frameworku ASP.NET MVC, jeho historie, využití a základních principů fungování.

ASP.NET MVC je framework určený k webovému vývoji od společnosti Microsoft, která se jeho vývojem zabývá od roku 2007. Jedná se o alternativní možnost využití ASP.NET, které bylo dříve využíváno především technologií WebForms. Rozdílem je především využití architektury Model View Controller (MVC).

Framework kombinuje efektivitu a čistotu architektury MVC, nejnovější techniky agilního vývoje a to nejlepší ze samotné platformy ASP.NET (Freeman, 2013). Od roku 2014 je framework navíc k dispozici jako open-source projekt, což dává všem vývojářům možnost podílet se na jeho vývoji (Galloway et al., 2014).

4.1 Architektura MVC

Pro označení MVC se používá také český ekvivalent MPO, který znamená Model – Pohled – Ovládání. „Stručná charakteristika vzoru: MPO (Model/ Pohled/ Ovládání) sestává ze tří druhů objektů. Model představuje objekt aplikace, Pohled prezentaci na obrazovce a Ovládání definuje způsob, jak uživatelské rozhraní reaguje na vstup od uživatele. MPO odděluje tyto objekty a tím zvyšuje flexibilitu a znovu použitelnost celého řešení“ (Pecinovský, 2007).

Pokud se na architekturu podíváme přímo z pohledu webového frameworku ASP.NET MVC, jsou funkce jednotlivých částí následující (Freeman, 2013).

- Modely reprezentují data, se kterými uživatel pracuje. V rámci frameworku pak máme modely dvojího typu. Prvním typem jsou jednoduché pohledové modely, které slouží pouze k reprezentaci dat, přenášených mezi Ovladačem a Pohledem. Druhým typem jsou doménové modely. Ty vznikají identifikací objektů reálného světa, jejich pravidel, operací nebo aktivit, známých také jako domény. Doménové modely tedy uchovávají data těchto objektů, ale také operace, transformace a pravidla pro jejich manipulaci.
- Pohledy jsou použity k renderování nějaké části modelu jako uživatelské rozhraní.

- Ovladače slouží k zpracování příchozích požadavků, vykonání akcí na modelu a k zvolení pohledů zobrazených uživateli.

Základní myšlenkou MVC architektury je oddělení logiky od výstupu. Řeší tedy problém tzv. "špagetového kódu", kdy máme v jednom souboru (třídě) logické operace a zároveň renderování výstupu (ITnetwork.cz, 2015). Obrázek 1 v kapitole 4.4 popisuje životní cyklus webové stránky ve frameworku ASP.NET MVC a je na něm mimo jiné zachycena komunikace jednotlivých částí architektury MVC.

4.2 Programovací jazyk C#

Jak již bylo napsáno výše, k vývoji ve frameworku ASP.NET MVC se využívá programovací jazyk C#. Jedná se o programovací jazyk vyvíjený firmou Microsoft, jehož první veřejně dostupná verze byla představena v roce 2000. Od té doby bylo vydáno šest verzí, poslední C# 6.0 je poté z roku 2015. *„C# (čteno "C sharp") je programovací jazyk, který je navržen pro vytváření různorodých aplikací, které běží na rozhraní .NET Framework“* (MSDN, 2016).

Ve specifikaci jazyka C# dle standartu ECMA[1] (Standard ECMA-334, 2006) jsou definovány následující cíle využité při jeho návrhu:

- C# je jednoduchý, moderní, univerzální, objektově orientovaný programovací jazyk,
- jazyk musí poskytovat podporu pro principy softwarového inženýrství, jako jsou silně typové kontroly, kontroly omezení, detekce použití neinicializované proměnné, garbage collection[2],
- jazyk je navrhnut pro použití při vývoji softwarových komponent vhodných pro nasazení v distribuovaných prostředích,
- důraz na portabilitu zdrojového kódu a jednoduchost přechodu stávajících programátorů (zvlášť těch, kteří jsou obeznámeni s jazykem C či C++),
- podpora pro internacionalizaci,

[1] ECMA (European Computer Manufacturers Association) je mezinárodní soukromá nevýdělečná organizace pro normalizaci informačních a komunikačních systémů s otevřeným členstvím.

[2] Garbage collection (v překladu „odvoz odpadu“) je způsob automatické správy paměti, kdy speciální algoritmus (garbage collector) vyhledává a uvolňuje úseky paměti, které již program nevyužívá.

- vhodnost využití jazyka pro hostovaný i vestavěný systém a to od velkých systémů, využívajících sofistikované operační systémy po velmi malé mající dedikované funkce,
- ačkoli mají být aplikace napsané v jazyce C# ekonomické vzhledem k paměti a vytížení procesoru, není jazyk vzhledem k výkonu a velikosti navržen k náhradě jazyka C či nízkoúrovňových jazyků.

Programovací jazyk C# má široké využití od tvorby databázových programů, webových aplikací a webových služeb po formulářové aplikace ve Windows či software pro mobilní zařízení.

4.3 Historie a vývoj frameworku

Jak již bylo řečeno, framework ASP.NET MVC pracuje na platformě ASP.NET. Ta je součástí frameworku .NET a byla ve verzi 1.0 vydána v roce 2002. ASP.NET je nástupcem technologie ASP (Active Server Pages), avšak podstatně se od ní odlišuje. Hlavní odlišnost je ta, že ASP.NET nevyužívá skriptovací jazyk (VBScript), nýbrž jazyky frameworku .NET (C#, VisualBasic), které jsou kompilované. *„Aplikace napsaná v ASP.NET se dynamicky zkompiluje ve chvíli, kdy k ní přistoupí klient. Kopie aplikace se uloží do cache pro potřeby budoucích požadavků. Pokud se některý ze souborů změní, aplikace se automaticky překompiluje, jakmile ji bude vyžadovat klient. To je zásadní rozdíl oproti skriptovacím jazykům, u kterých jsou stránky při každém přístupu klienta znovu a znovu parsovány. Aplikace v ASP.NET je proto znatelně rychlejší“* (Podešva, 2011). Dalšími rozdíly jsou např. plná podpora objektově orientovaného návrhu či možnost využití ovládacích prvků a knihoven tříd převzatých právě z frameworku .NET (MSDN, 2016). Díky ASP.NET tak byl výrazně snížen rozdíl, mezi vývojem klasických desktopových aplikací a webovým vývojem.

Společně s ASP.NET byl v představen i framework ASP.NET WebForms. Cílem technologie bylo odstranit omezení bezstavového webového prostředí, čehož bylo docíleno pomocí HTML a JavaScriptu, použitím ViewState a Session (Podešva, 2011). WebForms svým způsobem umožňují vyvíjet webovou aplikaci podobně, jako desktopovou; webové stránky se skládají z ovládacích prvků podobným těm v operačním systému Windows (tlačítka, popisky, textová pole atd.). Tyto ovládací prvky v sobě mají implementovanou schopnost pro zobrazení využívajíce opět technologií HTML a JavaScript. Základem tvorby webové aplikace ve WebForms je poté řízení aplikace událostmi, které nastávají obvykle určitou uživatelskou akcí – klik či pohyb myší nad daným ovládacím prvkem, stisk tlačítka atd. Princip implementace webové aplikace

v technologii WebForms je tak velice podobný implementaci desktopové aplikace s GUI[3] (ať už v technologii WinForms, která využívá i stejného programovacího jazyka a knihoven, tak Java). Výhodou WebForms je zcela jistě rychlost tvorby, zejména aplikací obsahujících velké množství formulářů. Nevýhodou je poněkud komplikovanější architektura a zvýšená velikost HTML stránky, kvůli odesílání ViewState.

Framework ASP.NET MVC byl poprvé představen v roce 2009. Využívá také běžných vlastnosti ASP.NET jako jsou např. uživatelské role, Sessions atd. Jak již bylo napsáno výše, jeho hlavní charakteristikou a také odlišností od WebForms je využití architektury MVC. Framework nevznikl jako náhrada WebForms, jedná se pouze o další alternativu vývoje webu. Kromě architektury má ASP.NET MVC ještě další zásadní odlišnost. *„Protože jsou logika a prezentace webové aplikace díky architektuře MVC odděleny a jednotlivé komponenty jsou na sobě de facto nezávislé, umožňuje ASP.NET MVC bez problémů paralelní vývoj aplikace, kdy se jeden programátor může zabývat Modelem, jiný Views a další Controllerem“* (Podešva, 2011). Tento fakt je poměrně zásadní a je využit v kapitole zabývající se úpravou metodiky MMSP. Další odlišností je vztah mezi URL a generovanou stránkou. URL v případě ASP.NET MVC neodkazuje na konkrétní stránku (soubor), uloženou na serveru ale určuje, jaký spustit ovladač a akci. Je pak již na logice ovladače zvolit, která stránka bude uživateli vrácena (Galloway et al., 2014).

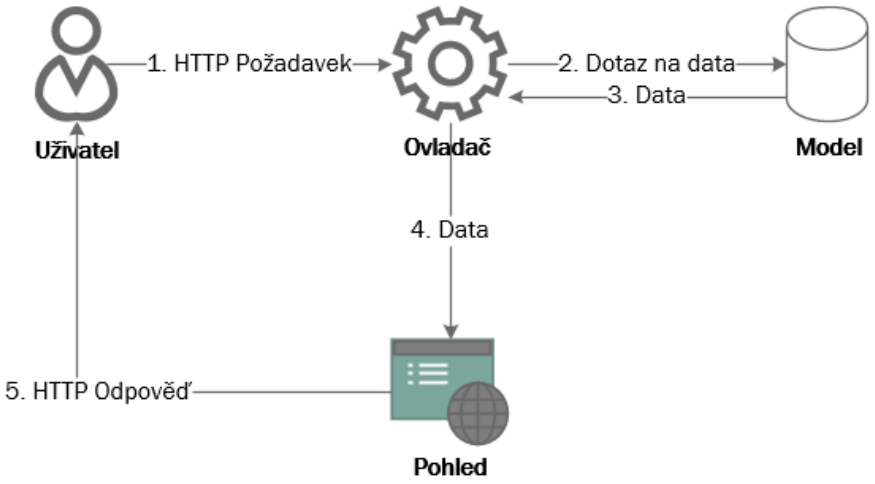
Mezi výhody ASP.NET MVC patří např. jednoduchost a efektivnost životního cyklu stránky (viz kapitola 4.4); využívání pouze HTML prvků, což usnadňuje ladění aplikace; možnost umístit na jednu stránku více formulářů a v neposlední řadě také snadnější využití novinek v oblasti webových technologií, jako je např. HTML5. Odlišností mezi technologiemi je mnoho a v rámci práce jsou některé popsány, primárně se však věnuji frameworku ASP.NET MVC.

Od představení první verze frameworku ASP.NET MVC bylo vydáno celkem 5 hlavních a několik minor (vedlejších) verzí. Je tak jasné, že framework prošel mnoha změnami, jejichž popis však přesahuje rozsah a zaměření této práce. Veškeré funkčnosti popsané v této práci se vztahují k poslední stabilní verzi ASP.NET MVC 5.2, vydané v únoru 2015.

4.4 Životní cyklus stránky

Průběh životního cyklu webové stránky ve frameworku ASP.NET MVC zachycuje následující obrázek.

[3] GUI – Grafické uživatelské rozhraní (z překladu Graphical User Interface).



Obrázek 1: Diagram životního cyklu načtení stránky v ASP.NET MVC

Vše začíná u uživatele, který zadá požadovanou adresu do webového prohlížeče, neboli vytvoří HTTP požadavek. Tím je vyvoláno směrování aplikace, které na základě konfigurace hledá odpovídající ovladač. Pokud ovladač neexistuje, vrací se uživateli chyba. V opačném případě se uvnitř ovladače najde požadovaná akce a dochází ke komunikaci s modelem, případně se provádí různé dotazy na filtrování dat apod. Data se vrátí do ovladače a ten zavolá příslušný view engine[4], který na základě pohledu dynamicky vygeneruje výslednou HTML stránku. HTTP odpovědí je následně uživateli tato stránka vrácena.

4.5 Práce s databází

Téměř žádná webová aplikace se neobejde bez databáze. ASP.NET využívá pro práci s databází technologii ADO.NET. Ta komunikuje s DBMS (Database management system) jako je SQL Server či Oracle. Pro usnadnění práce s databází se v rámci vývoje (nejen) webových aplikací v ASP.NET MVC využívají tzv. objektově-relační mapovací frameworky, nejpoužívanějším je Entity Framework. Tyto frameworky poskytuji rozhraní pro snadnější používání ADO.NET, mapují tabulky databáze na objekty a usnadňují psaní SQL dotazů. Entity Framework v počáteční inicializaci nabízí hned několik přístupů pro práci s databází. Ať si však programátor zvolí přístup jakýkoli, vždy je třeba mít v nějaké formě zpracovaný návrh databáze (MSDN, 2016).

[4] View engine je zodpovědný za vytváření HTML kódu na základě definovaných Pohledů. Od verze ASP.NET MVC 3 se obvykle využívá Razor engine. Je však možné využít i WebForm engine a další.

4.6 Vývojové prostředí

K vývoji webových aplikací v ASP.NET MVC se využívá vývojové prostředí (IDE) zvané Visual Studio vyvíjené společností Microsoft. Visual Studio obsahuje kompletní sadu nástrojů pro vývoj na platformě ASP.NET a to nejen pro webové aplikace. Jeho omezení spočívá v dostupnosti pouze pro operační systém Windows.

Největší výhodou tohoto prostředí je tzv. IntelliSense, umožňující automatické doplňování příkazů. Technologie dále poskytuje výpisy s informacemi o třídách, či metodách, se kterými vývojář pracuje a to nejen v jazyku C#, ale i při úpravě HTML, CSS stylů a JavaScriptu. Další velmi užitečnou funkční prostředí je nástroj pro ladění (debugging), který umožňuje zastavit program při jeho vykonávání a prohlédnout si zavedené proměnné či zpracovávat kód po jednotlivých řádcích. Visual Studio obsahuje mnoho integrovaných funkcí, umožňujících např. tvorbu databázových schémat, správu verzí, kontextovou navigaci, automatické hledání chyb a možností řešení, refaktoring nebo návrh designu. Kromě toho lze do IDE přidávat rozšíření obsahující dodatečné funkčnosti na téměř každé úrovni (MSDN, 2016).

Visual Studio je aktuálně vydávané v třech edicích, přičemž většina potřebných funkcí je obsažena v edici Community (Dříve Expess), dostupné zdarma. Oproti edici Professional či Enterprise má menší podporu pro týmovou spolupráci kvůli absenci TFS[5] a neobsahuje nástroje pro testování výkonu, zatížení či profilování (umožňuje pouze jednotkové testování). Tyto nedostatky lze však nahradit programy třetích stran a v malém týmu lze tak zcela bez problémů (i z právního hlediska) využit edici Community.

4.7 Hlavní výhody frameworku

V této kapitole jsou shrnuty nejpodstatnější výhody frameworku ASP.NET MVC, díky kterým se platformě ASP.NET podařilo odstranit nedostatky WebForms a umožnilo jí konkurovat nejmodernějším technologiím, jako např. Rails.

4.7.1 Využití MVC architektury

Interakce uživatele s MVC aplikací sleduje přirozený sled událostí. Na základě akce od uživatele aplikace mění svůj datový model a vrací mu aktualizovaný pohled. Následně se

[5] TFS (Team Foundation Server) je nástroj pro týmovou spolupráci od společnosti Microsoft. Nejedná se pouze o nástroj pro správu kódu, nýbrž o nástroj podporující životní cyklus celého vývoje.

tento cyklus opakuje. Tato skutečnost dělá, vzhledem k povaze webových aplikací, z architektury MVC velice vhodnou volbu pro jejich vývoj (Freeman, 2013).

Webová aplikace je nutně tvořena několika technologiemi (např. databáze, HTML, spustitelný kód), obvykle rozdělených do několika vrstev. Tyto vrstvy do architektury MVC dobře zapadají a využívají jejích konceptů.

4.7.2 Rozšiřitelnost

Framework ASP.NET MVC je tvořen na sobě nezávislými komponentami, implementujícími rozhraní .NET. Tyto komponenty, mezi které patří např. směrovací systém či *view engine*, mohou být snadno nahrazené vlastní implementací. Obecně se pro každou komponentu nabízejí tři možnosti (Freeman, 2013):

- využití výchozí implementace (mělo by být dostatečné pro většinu aplikací),
- odvodit podtřídy výchozí implementace pro vyladění potřebného chování,
- nahradit komponentu novou implementací daného rozhraní.

4.7.3 Testovatelnost

Vysokou míru testovatelnosti a udržovatelnosti aplikace zajišťuje už samotná architektura MVC, která odděluje různé aplikační funkčnosti do nezávislých částí. Framework ASP.NET MVC však navíc využívá komponentově orientovaného návrhu s důrazem na to, aby každá jeho oddělená část splňovala požadavky pro jednotkové testování a tzv. mock[6] nástroje (TechNet, 2016). Pro jednotkové testy je možné využívat buď přímo nástroje integrované ve vývojovém prostředí Visual Studio, nebo využít jiné, open-source nástroje, jako jsou např. NUnit či xUnit.

Kromě podpory jednotkových testů je framework uzpůsoben práci s nástroji pro automatizaci testování uživatelského rozhraní. Je možné psát testy simulující interakci uživatele, a to nezávisle na konkrétních HTML strukturách, či CSS třídách.

[6] Mock je nazýván objekt, využívaný při testování, který zastoupí některé skutečné třídy a umožní se tak v rámci testu soustředit plně na testovanou logiku.

4.7.4 Ostatní

Mezi další výhody frameworku, které považuji za důležité zmínit, patří jeho směrovací systém. Díky němu je snadné tvořit čisté a srozumitelné URL adresy, které neodkazují přímo na soubory na disku a jsou tak nezávislé na své implementaci. Mimo jiné jsou i lépe podporovány vyhledávacími systémy.

Jak již bylo řečeno, framework ASP.NET MVC je open-source, což znamená, že je možné stáhnout originální zdrojové kódy, modifikovat je a kompilovat. Jedinou výhodou však není možnost modifikace kódu, ale také lepší pochopení, jak jsou různé komponenty implementovány a jak fungují. Součástí zdrojových kódu jsou i původní komentáře od vývojářů.

Za velkou výhodu frameworku lze také považovat využití osvědčených komponent a možností vývoje samotné platformy ASP.NET. Díky tomu je umožněno psaní kódu v jakémkoli jazyce .NET, či využívání celého .NET ekosystému včetně knihoven třetích stran. Zároveň jsou v rámci ASP.NET již naimplementovány často využívané funkčnosti jako přihlašování, správa rolí, profilů, či podpora pro internacionalizaci (TechNet, 2016).

5 Návrh metodiky MMSP – ASP.NET MVC

Stěžejní kapitola práce, zabývající se samotnou úpravou zvolené metodiky MMSP pro vývoj webových aplikací ve frameworku ASP.NET MVC. Jednotlivé úpravy jsou zde řazeny do podkapitol, zvolených především pro zachování struktury původní metodiky. Nově přidané kategorie a principy jsou vždy pojmenovány podle konvencí, odpovídajících architektuře UMA, případně lokalizací uvedených v diplomové práci zabývající se použitím nástroje EPFC (Šimko, 2015).

V rozšiřování metodiky MMSP vycházím z poznatků získaných z literatury uvedené ve zdrojích, best-pratice a vlastní praxe. Verze metodiky, ze které vycházím, je poslední verze MMSP 1.5 zpracovaná pomocí nástroje Eclipse Process Framework Composer (MMSP, 2011).

V následujících podkapitolách jsou postupně navrhnuty změny v rámci jednotlivých prvků metodiky MMSP, představených v kapitole 3.3. Konkrétně se jedná o úpravu Rolí, Úloh, Pracovních produktů, Životního cyklu metodiky a Návodů. V poslední podkapitole je pak popsán proces publikace metodiky MMSP – ASP.NET MVC pomocí nástroje EPFC.

5.1 Role

Jak bylo popsáno v kapitole 3.3, metodika MMSP definuje šest rolí - Projektový manažer, Analytik, Tester, Vývojář, Architekt, Zainteresovaná strana a roli Nedefinovaná, která je určena pro všechny ostatní role. „U každé role je popsán účel role v projektovém týmu, požadované schopnosti a dovednosti, úlohy a pracovní produkty, za které je zodpovědná“ (Buchalcevová a Stanovská, 2013).

V rámci úpravy rolí jsou především specifikovány role, které by v metodice MMSP patřily pod obecnou roli Vývojář, jejíž úlohy a pracovní produkty je možné vidět na následujícím obrázku..

Obrázek 2: Úlohy a pracovní produkty role Vývojář (zdroj: MMSP, 2011)

Nově tak byly do metodiky přidány role Návrhář UI, Vývojář UI, Databázový specialista a Programátor. Tyto role jsou podrobněji definovány v následujících podkapitolách ve struktuře odpovídající metodice MMSP.

5.1.1 Návrhář UI

První rolí definovanou v rámci rozšíření metodiky MMSP je návrhář uživatelského rozhraní, nebo také návrhář UI (z anglického User Interface).

Obrázek 3: Úlohy a pracovní produkty role Návrhář UI

Návrhář UI je v týmu společně s Architektem zodpovědný za strukturu webové aplikace z hlediska prezentační vrstvy. Vychází ze specifikace požadavků, kde jsou definovány funkce a možnosti, jež by měla aplikace a její GUI pokrývat. Jeho úkolem je návrh rozložení navigace, specifikace webových prvků, jejich umístění a definice interakcí mezi jednotlivými funkcemi. Na základě těchto poznatků následně vytváří celkový pohled ve formě drátěného modelu uživatelského rozhraní, neboli wireframe, ve kterém je definováno rozmístění funkčních prvků na webových stránkách.

Kromě struktury webové aplikace zpracovává návrhář UI i design webové aplikace. To zahrnuje především zpracování vzhledu jednotlivých komponent stránky, právě podle vytvořených wireframů. Připravuje tak např. použitá barevná schémata, typy písem a homogenní vzhled všech použitých komponent (GML, 2014).

Požadované dovednosti

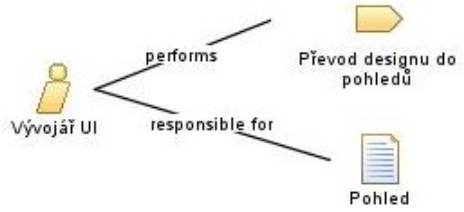
- Znalost nástroje na tvorbu návrhu UI
- Zkušenosti z oblasti návrhu UI
- Zaměření na použitelnost řešení
- Alespoň zběžná znalost možností technologií, ve kterých se vývoj provádí
- Kreativita a estetické cítění k vizuálnímu pojetí aplikace
- Smysl pro detail

Přiřazení role

V případě menších projektů je možné, aby tuto roli zastával jeden z analytiků. Návrhář UI musí brát v potaz spíše využití a funkcionalitu daného řešení, nikoliv stránkou jeho vývoje (RUP, 2006) a je tak nevhodné, aby roli zastával Programátor. Činnosti role lze v případě potřeb rozložit do dvou rolí – Návrhář UI a Designér, jelikož se však metodika zabývá menšími projekty, jsou tyto role spojeny.

5.1.2 Vývojář UI

Jak z názvu role vypovídá, je zodpovědná za vývoj uživatelského rozhraní webové aplikace. Její činnost přímo navazuje na návrháře uživatelského rozhraní.



Obrázek 4: Úlohy a pracovní produkty role Vývojář UI

Úkolem role je zajistit převod grafického návrhu aplikace do formy, ve které bude na webu zobrazen. Jedná se tedy o převod do standardizovaných webových technologii HTML + stylování pomocí CSS.

Framework ASP.NET MVC označuje tyto HTML šablony jako tzv. Pohledy (views). Při jejich tvorbě umožňuje využití různých podpůrných metod, díky kterým je implementace UI rychlejší, efektivnější a usnadňuje budoucí změny.

Požadované dovednosti

- Znalost HTML, CSS
- Orientace ve frameworku ASP.NET MVC a znalost HTML Helpers, C#
- Zkušenosti z oblasti vývoje UI
- Schopnost komunikovat a řešit s ostatními členy týmu otázky návrhu a vývoje

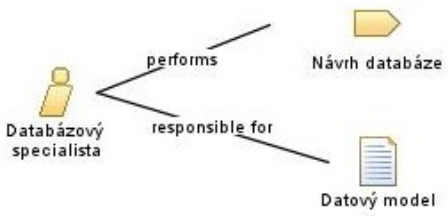
Přiřazení role

Osoba obsazená do této role musí komunikovat se všemi členy týmu, především pak s Programátorem, se kterým musí v průběhu vývoje úzce spolupracovat. Programátor se stará o veškerou logiku aplikace a dává Vývojáři UI všechna data ve formě vhodné pro

zobrazení. V případě menšího projektu je možné činnosti role Návrhář UI přenést právě do role Programátor.

5.1.3 Databázový specialista

Osoba zastupující tuto roli je zodpovědná za návrh databáze pro webovou aplikaci.



Obrázek 5: Úlohy a pracovní produkty role Databázový specialista

Databázový specialista je zodpovědný za definici detailního databázového návrhu zahrnujícího tabulky, atributy, indexy, pohledy, omezení, triggery a další specifické konstrukty pro schopnost ukládání, čtení, editaci a mazání perzistentních dat (RUP, 2006).

Rozsah úkolů vykonávaných databázovým specialistou se liší v závislosti na velikosti a složitosti vyvíjené aplikace a zvoleném typu struktury pro ukládání perzistentních dat (typicky relační databáze).

Požadované dovednosti

- Znalost datového modelování a návrhu databáze
- Znalost Microsoft SQL Server a nástroje pro datové modelování
- Analytické a logické myšlení
- Komunikační dovednosti
- Optimalizace výkonu databáze

Přiřazení role

V případě malého týmu může být role zastoupena Programátorem, či jinou rolí mající potřebné znalosti.

Osoba zastávající roli databázového specialisty by měla být v ideálním případě zapojena i do nějakých dřívějších etap projektu, např. v analýze funkčností či požadavků (RUP, 2006).

5.1.4 Programátor

Role Programátor je zodpovědná za tvorbu programového kódu webové aplikace.

Obrázek 6: Úlohy a pracovní produkty role Programátor

Programátor vytváří modely, ovladače a ostatní komponenty tvořící spustitelnou aplikaci. Zároveň tato role zajišťuje psaní Unit testů a integraci všech částí webové aplikace (včetně částí vytvářených jinými rolemi).

Požadované dovednosti

- Dobrá znalost a zkušenost s frameworkem ASP.NET MVC a jazykem C#
- Znalost Javascriptu pro programování klientské části webové aplikace
- Zběžná znalost HTML a CSS
- Schopnost komunikovat, pracovat s ostatními členy týmu a koordinovat
- Znalosti jazyka UML pro pochopení jednotlivých modelů vytvořených Analytikem
- Zkušenost s tvorbou testů, které pokrývají očekávané chování technických komponent systému

Člen týmu v roli Programátora by měl především disponovat perfektními znalostmi technické oblasti, které se na projektu věnuje. „Zároveň by však měl alespoň všeobecně rozumět všem technologiím, které jsou na projektu využívány. Jeho přehled a znalosti jiných oblastí, než kterým se detailně věnuje, usnadňují jeho komunikaci s ostatními členy týmu“ (MMSP, 2011).

Přiřazení role

V roli Programátora může vzhledem k rozsahu vystupovat více lidí. Ve velmi malých agilních týmech mohou jako Vývojáři vystupovat i osoby, kteří již mají přidělenou roli jinou (typicky Vývojář UI/Databázový specialista, ale často je to i Analytik).

U větších projektů by bylo vhodné z role Programátor separovat úlohy integrace a přiřadit je nové roli Integrátor.

5.2 Úlohy

Kapitola se zabývá popisem nových a upravených Úloh metodiky MMSP, definovaných v rámci kapitoly 3.3, přesněji na Úlohy disciplíny Architektura a Vývoj. V metodice MMSP se úlohy disciplíny Architektura omezují pouze na rozpracování architektury a návrh architektury. V případě disciplíny Vývoj pak na návrh řešení, implementaci řešení, tvorbu a provádění unit testů, integraci a vytvoření buildu. Z hlediska vývoje webových aplikaci, specificky pak v ASP.NET MVC lze identifikovat úloh více. Následující podkapitoly uvádějí úlohy a jejich náplně z oblasti architektury a vývoje od návrhu UI po kompletní implementaci.

Nově identifikované úlohy vzniklé doplněním metodiky MMSP jsou popsány níže v separátních podkapitolách. Struktura každé z podkapitol vychází ze struktury upravované metodiky a je následující:

- popis a účel úlohy,
- přiřazení k disciplíně,
- vstupy a výstupy úlohy,
- role vykonávající úlohu,
- potřebné kroky k vykonání úlohy.

5.2.1 Úloha Tvorba wireframe

Účelem úlohy je vytvoření wireframu pro všechny hlavní stránky tvořící aplikaci a udržením tak konzistence mezi nimi. Wireframe představuje základní zobrazení všech komponent webové stránky a jejich vzájemné umístění a je ho možné využít jako podklad pro návrh designu či pro získání zpětné vazby od zákazníka. Úloha je zařazena do disciplíny architektura a vykonává ji Architekt a Návrhář UI.

Vstupem úlohy jsou funkční i nefunkční požadavky, Vize, Slovník pojmů, zpracované Případy užití. Výstupem je poté produkt Wireframe.

Kroky úlohy

1. Stanovení počtu vytvářených wireframů

   Pro menší, nebo méně komplexní webové stránky je typicky dostačující jeden wireframe sloužící jako šablona pro všechny vyvíjené obrazovky. Pro komplexnější projekty je třeba wireframů sestavit více, jejich počet se odvíjí od funkční a navigační rozmanitosti produktu (Garret, 2011).

2. Určení způsobu reprezentace wireframu

   Pro menší projekty je dostačující náčrt tužkou, případně je možné využít specializovaný software. Záleží také na stanoveném počtu vytvářených produktů z předchozího kroku. Seznam možných způsobů realizace je v kapitole 5.3.1, popisující samotný produkt.

3. Tvorba produktu

   Při samotném návrhu wireframe je doporučeno držet se osvědčených postupů, popsaných v rámci Návodu (kapitola 5.5.2).

5.2.2 Úloha Návrh grafického designu

Účelem úlohy je navrhnutí uživatelského rozhraní aplikace, včetně struktury navigace mezi jednotlivými stránkami webu. Design musí odpovídat požadavkům, musí mít logické uspořádání a především musí umožňovat dobrou použitelnost webu. Úloha je zařazena do disciplíny Vývoj a vykonává ji Návrhář UI společně s Analytikem.

Vstupem úlohy jsou funkční i nefunkční požadavky, Vize, Slovník pojmů, zpracované Případy užití a Wireframe. Výstupem je poté Navigační mapa a zpracovaný Design webové aplikace.

Kroky úlohy

1. Analýza charakteristik cílových uživatelů

   Uvědomit si, kdo jsou uživatelé systému a pokusit se analyzovat, jak jsou zvyklí s webem pracovat a jak budou novou aplikaci používat. Zaměření na jednoduchost a použitelnost.

2. Identifikování hlavních stránek a elementů uživatelského rozhraní

   Identifikace stránek a jejich elementů, které budou uživatelem zobrazeny vždy či velice často. Jedná se o úvodní stránku, která je zobrazena při vstupu na web, dále o navigaci a stránky tvořící jádro aplikace, na kterých bude uživatel trávit nejvíce času.

3. Definování navigační mapy

   Vytvoření mapy webové stránky - přehled všech důležitých stránek a přechodů mezi nimi. Nemusí obsahovat všechny možné přechody, stačí hlavní (RUP, 2006).

Za počáteční prvek mapy je stanovena úvodní stránka, následně jsou na ni navazovány ostatní stránky aplikace do hierarchie reprezentující počet kliků k jejich dosažení.
V průběhu tvorby navigační mapy je důležité myslet na snížení tohoto počtu kliků pro často navštěvované stránky. Je však nutné myslet i na přehlednost a zachování srozumitelné struktury webu a nesnažit se tak toto číslo minimalizovat za každou cenu.

4. Návrh elementů uživatelského rozhraní

Definují se velikosti, barvy, styly a chování (při najetí myši, kliknutí atd.) jednotlivých elementů, ze kterých je web tvořen.
Důležité je identifikovat společné prvky, se kterými se uživatel při práci s aplikací bude setkávat a jednoznačně jim definovat vzhled a chování (RUP, 2006). Jedná se samozřejmě o navigační lištu, ale také jiné prvky umístěné na více stránkách – např. tlačítka s podobnou funkcionalitou, grafy, tabulky, vyhledávání a způsob zobrazení (řazení záznamů v tabulce dle názvu). Všechny tyto elementy musí být v rámci všech stránek stejně jednotně navrženy. Uživatel se poté naučí tyto *vzory* a dokáže podle části aplikace lépe obsluhovat i zbytek.

5.2.3 Úloha Převod designu do pohledů

Účelem úlohy je implementace uživatelského rozhraní aplikace. Jednotlivé obrazovky jsou reprezentovány tzv. Pohledy, které jsou v rámci této úlohy vytvářeny. Pohledy vycházejí z navrhnutého Designu webové aplikace, přičemž by změny vzhledu měly být minimální a odůvodněné. Úloha je zařazena do disciplíny Vývoj a vykonává jí Vývojář UI společně s Programátorem. V případě potřeby je vhodná spolupráce s Analytikem či Návrhářem UI.

Při implementaci je důležité dbát na to, aby všechny formuláře a prvky napříč webovou aplikaci byly z hlediska vzhledu konzistentní. V případě že aplikace využívá nějaké hotové komponenty (datová tabulka, galerie atd.), je třeba je také pomocí CSS vhodně stylizovat. Pohledy je vždy důležité zkontrolovat ve všech hlavních prohlížečích (i jejich starších verzích) – každý prohlížeč může stránku jinak vykreslit, nebo dokonce změnit funkčnosti.

Vstupem úlohy jsou funkční i nefunkční požadavky, Slovník pojmů a Design webové aplikace. Výstupem jsou poté jednotlivé Pohledy.

Kroky úlohy

1. Identifikovat použité komponenty

Analýza Designu webových stránek a identifikování jednotlivých komponent. Návrh způsobu realizace těchto komponent v Pohledu. Jedná se všechny prvky použité na webu od vstupních polí, rozbalovacích seznamů, po datové tabulky, grafy, galerie apod.

2. Návrh layoutu stránek

   Layout pomáhá k zachování konzistentního vzhledu napříč pohledy (Freeman, 2013). V tomto kroku je třeba určit, které části webových stránek se budou často opakovat – obecně se jedná o hlavičku, navigaci apod. Z nich se následně vytvoří layout, který bude součástí více pohledů. Výhodou je mimo jiné i snadná úprava – vše je definováno na jednom místě.

3. Tvorba Pohledů

   Následuje samotné vytváření pohledů. To je realizováno kódem HTML a kaskádových stylů. Pro usnadnění je ve frameworku ASP.NET MVC dostupný tzv. Razor engine, umožňující generování HTML na základě C# kódu. Spolu s ním je možné využít i HTML helpery, díky kterým není třeba psát všechny komponenty ručně.
   Pohled obsahuje definované komponenty a statická data. Dynamická data jsou mu předávány ovladačem. Každému pohledu je možné přiřadit jeden model, který reprezentuje typicky nějakou entitu z reálného světa (zákazník, objednávka atd.) a jeho data jsou většinou uložená v databázi. Takto předaná data podléhají silné typové kontrole. Další možností, jak předat pohledu data je dynamická proměnná ViewBag (jejíž data jsou ukládána do slovníku ViewData), která je také předána ovladačem, nicméně do ní lze uložit cokoliv - její kód je vytvářen až po předání dotazu ovladači a není tak třeba předem definovat strukturu (Freeman, 2013).

4. Ověření na vzorových datech

   Aby se dal Pohled považovat za hotový, je třeba v něm zobrazit reálná data. Mnohdy s nimi může stránka vypadat jinak, zároveň se tak doladí různé parametry, aby výsledný produkt vypadal co nejlépe.

   V případě, že ještě není vytvořená logika aplikace, je vhodné otestovat Pohled na datech vzorových. V příslušných ovladačích se tak pouze pošlou do Pohledů vzorová data pomocí modelu či ViewBag. Tím se ověří celkový vzhled Pohledu, včetně částí psaných v C#.

HTML helpers

Jak již bylo zmíněno, HTML helpers jsou metody sloužící k zjednodušení implementace pohledů. Pomocí C# kódu je při zobrazení stránky generován přímo kód HTML.

Ačkoli by se mohlo zdát, že je jejich použití zbytečné a výsledek je stejný, jako v případě psaní HTML elementů přímo do editoru, není tomu tak. Neslouží totiž pouze k vytvoření HTML značek, ale i k zachování konzistence v celé aplikaci. Pomáhají zajistit, aby jednotlivé URL v pohledech směřovaly na správné umístění, aby formulářové elementy odpovídaly správným názvům a hodnotám využitého modelu a aby byly zobrazeny odpovídající chyby v případě vyplnění špatné hodnoty (Galloway et al., 2014).

Výhodou HTML helpers je automatické kódování hodnot, které jsou výstupem nějakého modelu. Pokud by se tedy v hodnotě objevil například řetězec `<br/>`, ve výsledném HTML kódu bude kódován na `&lt;br /&gt;`.

U všech HTML helpers je mimo jejich specifické atributy možné zadat přímo HTML atributy jako je třída, id apod. (TechFunda™, 2014).

HTML helpers jsou využívané především ve webových formulářích – jedná se o jeden z nejdůležitějších prvků webových stránek. Bez formulářů by byl web pouze statický, nebylo by možné vyhledávat či zadávat vstup jakoukoli formou (Galloway et al., 2014). Mezi tyto HTML helpers patří následující:

- Html.BeginForm – definuje HTML formulář. Vstupními parametry je název Ovladače a Akce, která bude formulář zpracovávat a typ metody odeslání (POST/GET).
- Html.ValidationSummary – Zobrazí validační chyby formuláře, které nejsou přiřazené ke konkrétní komponentě.
- Html.TextBox a Html.TextArea – Vytvoří vstupní pole. V případě textové oblasti je možné do parametrů definovat i její velikost.
- Html.Label – Vytvoří HTML značku *label* sloužící k asociaci popisku a vstupu.
- Html.DropDownList a Html.ListBox – Vytvoří výběrový seznam pro vstupní hodnoty. DropDownList umožňuje výběr jedné hodnoty a ListBox jedné či více. Hodnoty zobrazené v tomto seznamu předává pohledu jeho ovladač pomocí vlastnosti ViewBag.
- Html.ValidationMessage – Slouží k zobrazení chybové zprávy pro konkrétní pole. Jako parametr se uvádí název příslušného pole, přičemž samotná zpráva je opět zpracovávána v ovladači.

- Dále Html.Password, Html.RadioButton, Html.CheckBox

Zmíněné HTML helpers existují ještě ve variantě pro silně typová data pohledu. Ta jsou pohledu předávána jako model. Jednotlivé názvy metod pak sufix *For* (např. Html.LabelFor). V jejich případě není třeba do parametru zadávat text s názvem elementu, stačí přímo odkázat proměnnou modelu pomocí lambda výrazu. Využití silně typových dat přináší výhody ve formě automatického doplňování a oprav, či snadného refaktorování (Galloway et al., 2014).

Existují také jiné HTML helpers, které neslouží pro formuláře, ale k vytvoření odkazů na zdroje v aplikaci. Jedná se např. o Html.ActionLink, který vytváří odkaz na konkrétní akci zadaného ovladače. Dané akci lze samozřejmě přidat i parametry, které pak budou v příslušné metodě ovladače k dispozici (např. ID uživatele). K vytvoření URL využívá speciální směrovací API, což je oproti ručnímu psaní odkazu nesrovnatelně rychlejší a umožní předejití chybám vzniklým z napevno definovaných odkazů.

Kromě předdefinovaných HTML helpers v rámci frameworku existuje také možnost implementace vlastních. V rámci vlastních HTML helpers je možné vytvářet v podstatě jakkoli složitý obsah od jednoduchých značek po navigační menu, či tabulky s databázovými daty. Ukázka takto vytvořeného HTML helper je představena v rámci kapitoly 6.2.7.

5.2.4 Úloha Návrh databáze

Účelem úlohy je popsat proces návrhu databáze pro ukládání perzistentních dat webové aplikace. „Při datovém modelování na základě znalostí o realitě navrhujeme struktury dat, do kterých se budou o této realitě ukládat záznamy. Je to činnost, která vyžaduje důkladnou analýzu skutečnosti, a schopnost převést získané představy do databázových struktur. V souhrnu se této činnosti říká návrh databáze“ (Palovská, 2011).

Úloha je zařazena do disciplíny Vývoj a vykonává jí Databázový specialista. V případě potřeby je vhodná spolupráce s Analytikem.

Vstupem úlohy jsou funkční i nefunkční požadavky, Případy užití, Slovník pojmů. Výstupem je Datový model.

Pro návrh databáze je využita metodologie, kterou vytvořil Michael J. Hernandez. Autor poměrně netradičně sloučil všechny techniky návrhu databáze do sedmi fází. Tato metodologie měla celosvětově velký úspěch a je vhodná jak pro návrh zcela nové databáze, tak pro modifikaci či analýzu již existující (Velemínský, 2014).

Při procesu návrhu je třeba postupovat velice systematicky a nepodcenit, či dokonce nevynechat žádnou z části metodologie. Provést návrh databáze dle určité metodologie jen na půl je stejně špatné, jako nepoužít ji vůbec (Hernandez, 2006).

Kroky úlohy

1. Formulace úkolů a cílů

   Formulací úkolu se rozumí formulace účelu databáze. Měla by být stručná a věcná. „Cíle úkolu jsou tvrzením, které reprezentuje obecné úkoly podporované daty uloženými v databázi“ (Hernandez, 2006, s. 100). V první fázi se tak shromažďují data potřebná k následnému návrhu a je zde třeba hlavně analýza požadavků a rozhovory se zadavatelem (v případě možnosti i budoucími uživateli).

2. Analýza existující databáze

   Následuje analýza současné databáze. Nemusí se jednat o databázi digitální, často se jedná i o formu papírovou. Analýza napomůže k odhalení případných vad struktur a zjištění, zdali databáze podporuje současné požadavky. Výstupem této fáze je seznam polí (atributů) uchovávaných v databázi.

3. Vytváření datových struktur

   a. Zřízení struktur a tabulek – Vytváření seznamu tabulek a přiřazení atributů. Důležité je dbát na správná pojmenování - jednoznačné a pro všechny srozumitelné názvy, využívající co nejmenšího počtu slov a nepoužívání akronymů či zkratek.

   b. Primární Klíče – pomáhají jasně identifikovat záznam v tabulce a zajišťují integritu na úrovni tabulky.

   c. Specifikace polí – zahrnuje mimo jiné název, popis, datový typ, délku, povolené znaky, formát a možnost ukládání prázdných hodnot.

4. Vztahy mezi tabulkami

   V tomto kroku dochází k vytváření logických spojení mezi tabulkami. To napomáhá k minimalizaci redundantních dat a je díky tomu možné v aplikaci načítat data najednou z více tabulek.

   Existují tři typy vztahů: 1:1, 1:N a M:N. Jsou rozlišeny jednak funkčností, ale i implementací. Nejčastější typ vztahu je 1:N *„Z pohledu integrity se jedná o nejdůležitější druh vztahu, protože umožňuje eliminovat duplicitní data a udržet redundantní data na absolutním minimu“* (Hernandez, 2006, s. 229). Při

implementaci vztahu 1:1 a 1:N se využívá propojení tabulek pomocí primárního a cizího klíče. V případě vtahu M:N se pak využívá vazební tabulky, která vztah rozdělí na dva typu 1:N.

U definovaných vztahů je dále třeba určit jejich charakteristiku. Jedná se o určení typu a stupně účasti (povinnost existence záznamu v protilehlé tabulce). Také se definují pravidla v případě smazání záznamu z rodičovské tabulky. Existuje několik možností, jak se s mazáním vypořádat – nejpoužívanější volbou je buď nemožnost smazání záznamu, pokud existuje související v dceřiné tabulce, či volba tzv. kaskády, kde jsou automaticky smazány záznamy v dceřiných tabulkách. Další možnosti je pak nastavení hodnoty `null`, implicitní hodnoty, nebo žádnou akci ani kontrolu neprovádět (Velemínský, 2014).

5. Určení a definování business pravidel

   Business pravidla jsou omezení vyplývající z analýzy pro atributy či vtahy. Některá pravidla lze implementovat na úrovni databáze – např. seznamy přípustných hodnot pro jednotlivá pole (obvykle řešeno pomocí číselníků). Jiná se musí zpracovávat až na úrovni aplikace – např. kontrola, aby zadávaná hodnota do jednoho sloupce byla vždy vyšší než ve sloupci druhém.

6. Určení a definování pohledů

   Pohled je virtuální tabulka, která se skládá z polí jedné, či několika tabulek databáze. Je vhodné ho využít, pokud aplikace často zobrazuje data z více tabulek, nebo je v aplikaci více uživatelů, kterým se mají data zobrazit jinak. Dále je možné v pohledu definovat tzv. vypočítaná pole, která slučují data z existujících tabulek podle zadaného vzorce. Tuto funkčnost je samozřejmě možné implementovat až na úrovni aplikace, obvykle je však implementace na úrovni databáze rychlejší.

7. Kontrola integrity dat

   Poslední fáze návrhu databáze, ve které je doporučeno přezkoumat ještě jednou celou integritu databáze. Je doporučen modulární přístup – postupně projít všechny části celkové integrity: integritu na úrovni tabulek, polí, vztahů a business pravidel (Hernandez, 2006).

5.2.5 Úloha Implementace webové aplikace

Úloha je definována v metodice MMSP pod názvem Implementace řešení. V rámci rozšíření je přejmenována a přizpůsobena fremeworku ASP.NET MVC. Úloha zahrnuje všechny činnosti spojené s tvorbou a modifikacemi zdrojového kódu vyvíjené webové

aplikace, ať už za účelem zapracování nové funkcionality či oprav chyb identifikovaných v předcházejících iteracích.

Úloha je zařazena do disciplíny Vývoj a vykonává jí Programátor. V rámci implementace je obvykle nutná spolupráce s Analytikem, Testerem, Vývojářem UI i Databázovým specialistou.

Vstupem úlohy jsou funkční i nefunkční požadavky, Případy užití, Slovník pojmů, Databázový model. Výstupem jsou implementované Modely, Ovladače a Kód webové aplikace.

Implementace webové aplikace, nebo některé z jejích dílčích kroků mohou být v průběhu jedné iterace opakovány i několikrát. Jednotlivé přírůstky, které jsou implementovány, by měly být co nejmenší, aby byla doba mezi vlastní implementací a testováním co nejkratší a testování a případná náprava zjištěných chyb nezabrala příliš mnoho času (MMSP, 2011).

Je vhodné, aby již před samotnou implementací existovaly testy, ve kterých je jednoznačně definováno správné chování vyvíjené části. Implementace by pak měla být testována ihned po dokončení (OpenUp, 2012).

Kroky úlohy

Následující kroky není nutné provádět v pořadí zde popsaném, často se jednotlivé kroky při implementaci překrývají.

1. Definice strategie implementace

   V tomto kroku je třeba rozhodnout, jakým způsobem budou jednotlivé komponenty webu implementovány. Některé mohou být implementovány například znovupoužitím již vytvořených komponent, pro jiné je třeba psát zdrojový kód od začátku. Často komponenty mohou být implementovány pomocí různých knihoven či zásuvných modulů třetích stran.

2. Vytvoření Modelů

   Je třeba identifikovat objekty využívané v aplikace. Pro každý vytvořit model a definovat jeho atributy. Naprostá většina se bude shodovat s entitami definovanými při návrhu databáze.

3. Vytvoření Ovladačů

   Pro usnadnění implementace Ovladačů existuje v ASP.NET MVC funkce zvaná Scaffolding. Ta na základě Modelů vytvoří Ovladače (případně i Pohledy), se základními funkčnostmi jako je výpis, zobrazení detailu, editace a mazání. Jedná

se jen o kostru – ta se však v aplikacích opakuje natolik často, že byla zmíněná funkce vytvořena. Následně je nutné vytvořené Ovladače upravit tak, aby implementovaly všechny potřebné funkčnosti.

Scaffolding samozřejmě není nutné využít a implementovat všechny části aplikace ručně. Zde závisí čistě na rozhodnutí Programátora.

4. Konfigurace směrování

   Framework ASP.NET MVC nabízí dvě možnosti, jak směrování implementovat. První, která je součástí frameworku od první verze, se nazývá tradiční směrování (z angl. traditional routing). Ve verzi 5 bylo přidáno tzv. atributové směrování. Tradiční směrování využívá konvencí a centralizace, je však méně konfigurovatelné - např. je možné nastavit, aby jednotlivé URL odpovídaly názvům Ovladačů a jeho akcím. V případě atributového směrování je nutné pro každý Ovladač směrování definovat. Jeho výhodou je jednoduchost a spojení směrovacích URL přímo s Ovladači.

   Zvolení způsobu směrování závisí opět na stylu a preferencích Programátora. Nabízí se také možnost jejich kombinace.

5. Implementace zbylých částí aplikace

   V tomto kroku je zahrnuta např. implementace klientské části aplikace, což ve většině případů znamená psaní JavaScriptu, či využití/modifikaci dostupných knihoven, v případě jejich využití dle analýzy z prvního kroku.

   Dále je v kroku zahrnuto psaní různých pomocných tříd, znovupoužitelných prvků, specifikace filtrů akcí[7] a bundleru, konfigurace aplikace, zabezpečení aplikace apod.

V průběhu implementace je samozřejmě možné, že budou identifikovány určité problémy a omezení, které je nutné vždy konzultovat s týmem. Jakékoliv změny, které by měly být na základě zjištěných informací provedeny, musí být konzultovány se zainteresovanými stranami a samozřejmě zaznamenány do Seznamu požadavků na změnu (MMSP, 2011).

[7] Filtr akce je atribut, který lze přiřadit k jednotlivým akcím nebo celému Ovladači a který ovlivňuje způsob vykonání této akce. Díky filtrům lze například definovat práva uživatelů pro spuštění jednotlivých akcí.

5.2.6 Úloha Tvorba unit testů

Úloha je již v metodice MMSP definována. V rámci této kapitoly je však rozšířena a přizpůsobena pro framework ASP.NET MVC a tvorbu testů v aplikace Visual Studio.

„Úloha tvorba unit testů zahrnuje identifikaci a přípravu jednotkových testů, přičemž stejně jako samotné řešení, všechny její dílčí kroky probíhají inkrementálně, podle toho, jak vzniká vlastní zdrojový kód vyvíjené aplikace“ (MMSP, 2011).

Obecné zásady při tvorbě unit testů (Galloway et al., 2014):

- Testovat malé části produkčního kódu („units“)
- Testovanou část izolovat od zbytku kódu
- Testovat pouze výstupy (test nemusí vědět, jak kód pracuje)
- Automatizované spouštění všech testů s bivalentní návratovou hodnotou – prošel/neprošel

Pro tvorbu jednotkových testů ve vývojovým prostředí Visual Studio je připraven framework Microsoft unit-testing. Při zakládání nového projektu je možné společně s projektem vytvořit i přidružený projekt jednotkového testování, který již bude obsahovat sadu výchozích testů. Ty slouží především jako vzor pro vlastní implementaci testů (Freeman, 2013).

Ve většině případů jsou Unit testy psány jako mechanismus k zajištění kvality softwaru, až po implementaci dané funkčnosti. K psaní testů tak vývojář využije znalostí produkčního kódu a cíleného chování. Nevýhoda tohoto přístupu spočívá v tom, že vývojář může snadno přehlédnout nějakou část napsaného produkčního kódu, zvláště pokud jsou unit testy psány s větším časovým odstupem, což je běžná praxe. Tento problém řeší praktika zvaná Programování řízené testy (Test-Driven Developement), při jejímž použití je nejprve specifikována funkcionalita a psány testy pro její ověření. Psaní kódu následuje až po vytvoření testů. (Galloway et al., 2014).

5.3 Pracovní produkty

Kapitola se zabývá popisem nových a upravených Pracovních produktů metodiky MMSP, definovaných v rámci kapitoly 3.3. Pracovní produkty jsou výstupy jednotlivých úloh, činností a procesů. Stejně jako úlohy z předchozí kapitoly, spadají následující produkty do disciplín architektura a vývoj.

Kromě vytvoření produktů nových byly některé produkty metodiky MMSP revidovány či odstraněny:

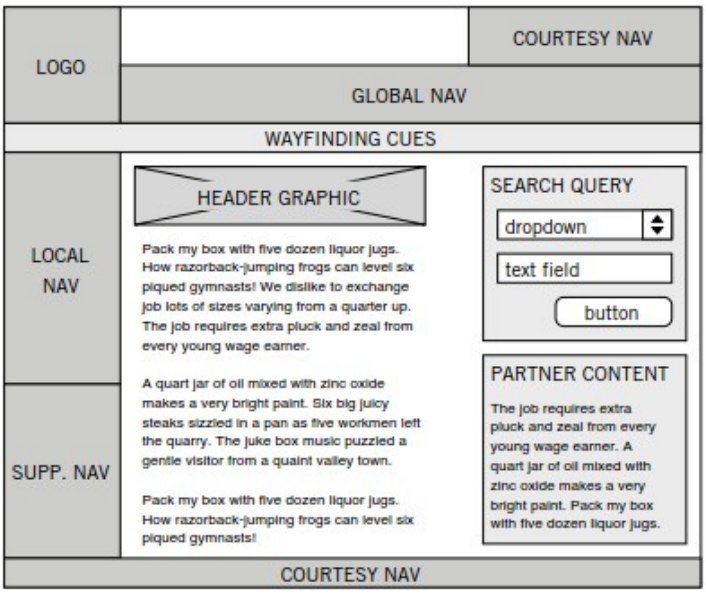
- Produkt Návrh byl vynechán, návrh je obsažen v jiných produktech definovaných níže.
- Produkt Build byl přejmenován na Build webové aplikace.
- Upraven produkt Unit test a Plán testů

Nově identifikované produkty vzniklé doplněním metodiky MMSP jsou popsány níže v separátních podkapitolách. Struktura každé z podkapitol vychází ze struktury upravované metodiky a je následující:

- Specifikace a účel produktu
- Zodpovědná role
- Důsledky nezhotovení projektu
- Formu, v jaké by měl být produkt vytvořen
- Podmínky, kdy není nutné pracovní produkt vytvářet

5.3.1 Wireframe

Wireframe (nazývaný také drátěný model) představuje základní zobrazení všech komponent webové stránky a jejich vzájemné umístění.



Obrázek 7: Wireframe webové stránky (zdroj: Garret, 2011)

Produkt integruje hlavní elementy struktury webových stránek (Garret, 2011):

1. Design uživatelského rozhraní – uspořádání a výběr prvků webové stránky.
2. Design navigace – identifikace a definice jádra navigačního systému aplikace.
3. Informační design – umístění a definice priorit informačních komponent.

Sloučením výše zmíněných návrhů do jednoho dokumentu je vytvořena jakási kostra webu, která staví na konceptuální struktuře a definuje cestu, kterou se bude ubírat následný vývoj. Součástí produktu jsou mimo zmíněných prvků také znění nadpisů, klíčových textů a tlačítek. Wireframe neobsahuje grafický návrh, je tvořen pouze pomocí čar a textu. Obecně se nepoužívají ani žádné obrázky či barvy.

Wireframe zpracovávají Architekt, Návrhář UI, může se na něm také podílet Analytik. Umožňuje získání zpětné vazby před realizací samotného designu, který z něj vychází.

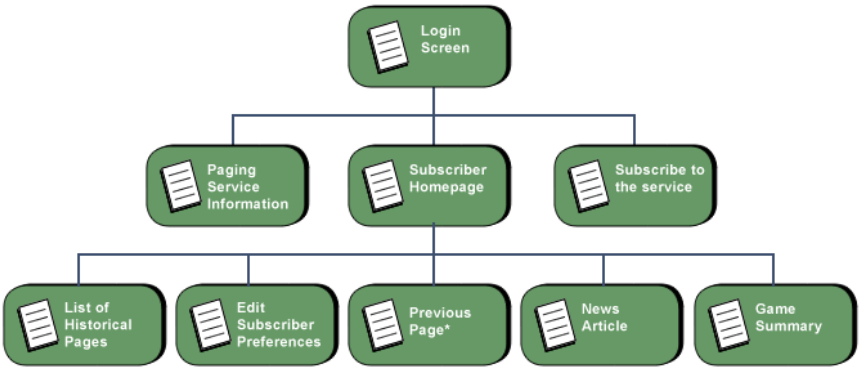
Tabulka 2: Produkt Wireframe

Důsledky nezhotovení produktu	Nekonzistence v uspořádání elementů na webových stránkách. Chybí zpětná vazba od zákazníka a prodlužují se následné implementační fáze.
Podmínky, kdy není nutné pracovní produkt vytvář**e**t	Wireframe je doporučeno v nějaké podobě zpracovat vždy.
Zp**ů**sob reprezentace	Wireframe může být reprezentován mnoha způsoby (Bank a Waleed, 2014): • Ruční náčrt + rychlost, flexibilita, ideální pro menší projekty - není interaktivní, nestandardizováno, spolupráce) • Papírové výstřižky + výstižnost, standardizace, interaktivita - rychlost, spolupráce • Textový či prezentační editor + obeznámenost, interaktivita - složitost návrhu, pouze základní elementy • Grafický editor + obeznámenost, výstižnost, podpůrné knihovny - množství funkcí editoru, spolupráce, interaktivita • Specializovaný software pro tvorbu wireframe + rychlost, široký výběr elementů, interakce, spolupráce - Nutnost naučit se s programem, limitované funkce

5.3.2 Navigační mapa

Navigační mapa, někdy také nazývaná jako mapa webu, slouží k zachycení hierarchie jednotlivých webových stránek. Jedná se o přehled všech důležitých stránek a přechodů mezi nimi. Není nutné, aby navigační mapa zobrazovala všechny přechody, ale jen ty hlavní (RUP, 2006).

Je využívána k ujasnění celého návrhu stránek v průběhu jejich návrhu. V nějaké formě se také často přidává i na samotný web, aby byla uživatelům k dispozici celková struktura stránek a tím usnadněna navigace.



Obrázek 8: Navigační mapa realizovaná stromovým diagramem (Zdroj: RUP, 2006)

Navigační mapa znázorňuje počet potřebných „kliků“ k přesunu na určitou stránku. Na základě tohoto čísla může být pak návrh upravován tak, aby důležité stránky byly viditelnější apod.

Navigační mapu zpracovává Návrhář UI, podílí se na ní také Analytik.

Tabulka 3: Produkt navigační mapa

Důsledky nezhotovení produktu	V případě, že navigační mapa není zhotovena, hrozí nepochopení celého konceptu webových stránek, případně špatná optimalizace z hlediska použitelnosti.
Podmínky, kdy není nutné pracovní produkt vytvářet	V případě malého počtu stránek aplikace (~5), jinak je doporučeno vytvářet produkt vždy.

Způsob reprezentace	Navigační mapu je možné zaznamenat v textové formě, avšak doporučuje se především využití grafického znázornění. To je možné reprezentovat více způsoby: • Hierarchický stromový diagram, kde každá úroveň stromu znázorňuje počet kliknutí k dosažení dané stránky (RUP, 2006). • Volný graf s vlastně definovanými ikonami – používán v případě, kdy je třeba do mapy zahrnout více vazeb, případně jiné skutečnosti.

5.3.3 Grafický design webové aplikace

Produkt slouží k popsání jednotlivých elementů, ze kterých budou webové stránky složeny. Umožňuje zkoumat a pochopit jejich návrh a připravit si ho dle představ bez nutnosti zasahování do zdrojového kódu (OpenUp, 2012). Design webových stránek vychází z wireframu, pokud byl zpracován. V opačném případě se wireframe vytváří společně s designem (Mittner, 2011).

V rámci produktu jsou definovány kódy použitých barev, typ a velkosti použitého písma, rozměry a chování (z hlediska UI) všech hlavních elementů, především těch, které se v aplikaci vyskytují opakovaně. Hlavním cílem tohoto produktu je zachování konzistence uživatelského rozhraní v rámci celé aplikace.

Design musí být vždy navrhován s ohledem na cílovou skupinu uživatelů dané aplikace. Měl umět uživatele zaujmout, avšak zůstat v pozadí a maximálně zpřehledňovat zobrazovaný obsah. Neměl by obsahovat zbytečné efekty, které by vedly k složité realizaci či nepřehlednosti. Musí působit přehledným a konzistentním dojmem.

Design webové aplikace zpracovává Návrhář UI, dále se na něm může podílet Analytik či Programátor.

Tabulka 4: Produkt Design webové aplikace

Důsledky nezhotovení produktu	Nekonzistentní rozhodnutí v oblasti grafického zpracování v průběhu vývoje webových stránek, což vede k menší použitelnosti.
Podmínky, kdy není nutné pracovní produkt vytvářet	Design webové aplikace je doporučeno v nějaké formě zpracovat vždy. Výjimkou může být případ, kdy vytvářená webová aplikace spadá pod jiný, již vytvořený web a bude jeho design zachovávat.

Způsob reprezentace	Design je obvykle reprezentován formou obrázku vytvořeného v grafickém editoru. Ten je následně rozřezán do vyhovujících celků a pomocí HTML + CSS převeden do formy webových stránek.

5.3.4 Pohled

Pohledy (z anligckého views) v rámci projektu ASP.NET MVC zajišťují prezentační vrstvu. „Jedná se o HTML šablonu, obsahující HTML stránku a tagy speciálního jazyka, který umožňuje do šablony vkládat proměnné, případně provádět iterace (cykly) a podmínky“ (ITnetwork.cz, 2015). Pohled lze považovat za „vizitku“ aplikace, pokud by byl design špatně zpracován, nebo by byl těžko použitelný, uživatelé nebudou chtít aplikaci využívat, třebaže jsou její zbylé části zpracovány bez chyb (Galloway et al., 2014) .

HTML v pohledech obecně definuje strukturu a obsah. Design stránky se poté vytváří pomoci kaskádových stylů (CSS). Výhodou je především to, že vzhled jednotlivých komponent, které se na webu opakují, je definován na jednom místě. Lze tedy poměrně jednoduše upravovat design celých stránek, aniž by se zasahoval do její struktury.

Kromě HTML a CSS je v pohledech využito také tzv. Razor engine (česky břitva), který slouží pro vkládání C# kódu do HTML. Břitva při načítání stránky zpracovává C# kód a na jeho základě generuje HTML, který je poslán do webového prohlížeče (Freeman, 2013). Použití Razor engine je možné vidět na následujícím výpisu – kód vypíše všechna jména osob v kolekci do odrážkového seznamu. Je zde vidět jednoduché propojení C# kódu právě s jazykem HTML.

Výpis 1: Příklad použití Razor engine v pohledu

```
@foreach(var person in personList) {<li>The name is @person.Name.</li>}
```

Jako Pohledy jsou označené veškeré obrazovky, které uživateli zobrazují nějaký výstup (většinou data jedné nebo více části nějakého Modelu) a je s nimi možné interagovat. Pohledy zajišťují pouze prezenční formu, neprobíhá v nich žádná úprava či tvorba dat. O to se starají ovladače (controllers). Součástí pohledů mohou být jednoduché podmínky – např. zobrazení tlačítka pro editaci objektu v případě, že má uživatel práva na editaci. Samotné ověření práv nicméně probíhá v ovladači, který pohledu dále předává pouze výsledek. Důležité je také zmínit, že Pohledy nejsou sami o sobě přímo přístupné, jako je tomu například u PHP. Do prohlížeče tak nelze zadat adresu pohledu a nechat ho vykreslit. Pohled je vždy vykreslován skrz ovladač, který mu také poskytuje data k zobrazení (Galloway et al., 2014).

Pohledů se vytváří v aplikaci mnoho. Dalo by se říct, že pro každou funkcionalitu je vytvořen pohled. Protože však ve většině případů zůstává základ stránky, jako je rozvržení, hlavička, navigace apod. stejný, je vhodné využít tzv. Layoutu. Na něm se definuje struktura, včetně navigace a ostatní pohledy se budou následně zobrazovat na vybraném místě (Freeman, 2013). Layout tak pomůže k zachování konzistentního vzhledu napříč pohledy.

Pohledy zpracovává Vývojář UI, především na základě produktu Design webové aplikace. Při jejich tvorbě musí spolupracovat s Programátorem a Návrhářem UI.

Tabulka 5: Produkt Pohled

Důsledky nezhotovení produktu	Chybějící prezentační vrstva – uživatel není schopen používat aplikaci.
Podmínky, kdy není nutné pracovní produkt vytvář**e**t	Pohledy je třeba vytvářet u každé webové aplikace vytvářené ve frameworku ASP.NET MVC.
Zp**ů**sob reprezentace	Kód HTML, CSS a C# uložený v souboru s koncovkou cshtml.

5.3.5 Datový model

Databázové model popisuje reprezentaci perzistentních dat využívaných webovou aplikací. Při vývoji webových aplikaci je využíváno především databází relačních a jinak tomu není ani v případě frameworku ASP.NET MVC.

Produkt definuje strukturu a formát dat relační databáze a vztahy mezi jednotlivými entitami. Slouží k vypořádání se s problémy sémantiky dat, konzistence dat, redundancí dat (Connolly et al., 2009) a k řešení persistence dat ve webové aplikaci.

Model by se měl vytvářet s ohledem na zvolený RDBMS (Relational database management system), což je v případě aplikací tvořených ve frameworku ASP.NET MVC typicky Microsoft SQL Server. V případě nutnosti je možné použití i jiného RDBMS, nemusí však být frameworkem plně podporován a práce s ním tak bude nejspíše komplikovanější.

Datový model může být vytvořen buď využitím zpětného inženýrství na existující databáze, nebo na základě přechozí analýzy požadavků (RUP, 2006).

Dle úrovně pohledu na data je možné dělit datové modely na několik typů (Connolly et al., 2009):

- Konceptuální model – nejvyšší úroveň, která zachycuje pouze třídy entit, jejich vztahy a atributy. Není ovlivněn budoucími prostředky řešení.

- Logický model – rozšiřuje konceptuální model z hlediska konkrétní databázové implementace (relační, objektové atd.).
- Fyzický model – nejkonkrétnější, popisuje fyzické uložení dat v konkrétním RDBMS.

Tabulka 6: Produkt Datový model

Důsledky nezhotovení produktu	Složitá modifikovatelnost databáze, malý výkon, nízká robustnost řešení.
Podmínky, kdy není nutné pracovní produkt vytvářet	Datový model nemusí být zapotřebí u webových aplikací pracující s malým množstvím perzistentních dat. Dále není datový model třeba, pokud se programátor rozhodne pro přístup *Code First*[8]. V takovém případě se tabulky databáze generují automaticky na základě tříd aplikace. I v tomto případě je však doporučeno zpracovat alespoň konceptuální model.
Způsob reprezentace	Datový model je typicky reprezentován UML diagramem.

5.3.6 Model

Modely (Models) jsou ve frameworku ASP.NET MVC chápány jako objekty používané k zasílání informací do databáze, k vykonávání výpočtů, validací a dokonce i k zobrazování dat v Pohledech. Jinými slovy tyto objekty reprezentují doménu, na kterou se aplikace soustředí a modely jsou objekty, které se mají v aplikaci zobrazovat, ukládat, vytvářet, upravovat a mazat (Galloway et al., 2014). Vytvořené Modely nejsou nic jiného než třídy napsané v jazyce C# reprezentující objekty, které jsou předmětem webové aplikace. Za produkt je zodpovědný Programátor.

Funkce modelu spočívá v přijetí parametrů zvenku a vydání dat ven. Model neví, odkud data v parametrech přišla a ani jak budou výstupní data zformátována a vypsána (ITnetwork.cz, 2015).

Důležitou funkcí modelu je také validace dat (vstupů) od uživatele. Vše je definováno anotacemi modelu a framework je využívá nejen k validaci, ale i k vytváření HTML kódu při zobrazení či editaci modelu. Anotace se přidávají přímo k definicím jednotlivých atributů, přičemž je možné definovat vlastní, nebo využít předdefinované (Galloway et al., 2014):

- Povinnost – definuje, zdali je nutné atribut ve formuláři vyplnit.

[8] Code First je jeden z přístupů inicializace databáze pomocí nejvyužívanějšího objektově-relačního mapovače zvaného Entity Framework

- Regulární výraz – omezení hodnoty na základě regulárního výrazu.
- Rozsah – minimální a maximální možné hodnoty pro číselné atributy.
- Porovnání – zajistí, aby měli dva atributy stejnou hodnotu (využívá se např. pro potvrzení emailu).
- Možnost validace na straně klienta s dotazem serveru (využívá technologie AJAX).
- Vlastní chybové zprávy v případě, že neprojde validace.
- Zobrazení – definuje jméno atributu, které je pak na stránce zobrazeno místo názvu atributu.
- Formát zobrazení – umožnuje definovat formát např. pro měnu či datum.

Takto definované validace je následně možné díky frameworku jednoduše aplikovat jak na klientskou, tak serverovou část webové aplikace. Validace na serveru je nutná vždy, neboť si uživatel může vynutit vypnutí klientské validace. Validace klientská nezbytná není, avšak v moderních webových aplikacích se jedná o nepsané pravidlo, zvyšující použitelnost aplikace.

Na následujícím výpisu je možné vidět příklad definovaného atributu s přidanými anotacemi.

Výpis 2: Definování atributu v modelu, včetně anotací pro validaci a zobrazení

```
public class UserModel
{
    [Display(Name = "Datum narození")]
    [DataType(DataType.Date)]
    [DisplayFormat(DataFormatString = "{0:dd. MM. yyyy}")]
    [Required(ErrorMessage = "Atribut {0} je povinný")]
    public DateTime BirthDate { get; set; }
}
```

Kromě využití předdefinovaných atributů pro validaci je možné v modelu implementovat i složitější validace vstupu, na základě nějaké logiky. Ukázka takto vytvořené validace je v kapitole 6.2.8.

Modely je možné využít jako základ celé webové aplikace. Framework ASP.NET MVC umožňuje díky funkci zvané Scaffolding vytvořit na základě definovaných modelů Pohledy a Ovladače, které jsou schopné základnách operací. Je tak automaticky vytvořena základní struktura webové aplikace, kterou je následně samozřejmě možné přizpůsobovat.

Tabulka 7: Produkt Model

Důsledky nezhotovení produktu	Webová aplikace není schopná komunikovat s databází.
Podmínky, kdy není nutné pracovní produkt vytvář**e**t	Produkt je nutné vytvořit v každé webové aplikaci.
Zp**ů**sob reprezentace	Každý model je reprezentován C# třídou.

5.3.7 Ovladač

Třetím prvkem, který osvětluje funkčnost celého vzoru je Ovladač (Controller). Jedná se o jakéhosi prostředníka, který zajištuje komunikaci mezi uživatelem, Modelem a Pohledem. Zodpovědnou rolí za produkt je Programátor.

Ovladače zajištují veškeré reakce na vstup od uživatele a občas dělají změny v Modelu. Jsou tak považovány za jakýsi „tok“ aplikace zpracovávající vstupní data a poskytující data zobrazená v relevantním Pohledu. ASP.NET MVC implementuje tzv. „přední ovladačovou variantu MVC vzoru“, což znamená, že je Ovladač v popředí každé vykonané události kromě směrování, viz 4.4 (Galloway et al., 2014).

Jako v naprosté většině částí webové aplikace psané ve frameworku ASP.NET MVC platí i v případě Ovladačů heslo „Konvence nad konfigurací“. Dodržení konvencí popsaných v kapitole 5.5.1 podstatně usnadňuje konfiguraci Ovladačů – zajistí spuštění odpovídající akce na základě podnětů od uživatele a zároveň vykreslení odpovídajícího pohledu, aniž by se musel explicitně specifikovat. To ušetří čas a především ulehčí jakékoli změny nejen u velkých webových aplikací. Např. navigace na stránku `/User/Details` ve výchozí konfiguraci spustí metodu `Details` v ovladači `UserController`. Pokud by se názvy nedodržely, bylo by nutné vytvořit explicitní směrovací pravidla.

Kromě názvu ovladače a akce je ještě možné v URL posílat parametry. Ty jsou v URL přidány za název akce, oddělené otazníkem. Např. je tak možné předat akci parametr jméno - `/User/List?Name=Jan`. V případě parametru ID se hodnota vkládá přímo do URL - `/User/Details/2`. V dané metodě jsou následně parametry přijaty jako deklarované argumenty a je možné je využít pro vrácení odpovídajícího Pohledu.

Nejčastěji existuje pro každou entitu (uživatel, produkt, objednávka atd.) právě jeden Ovladač.

Tabulka 8: Produkt Ovladač

Důsledky nezhotovení produktu	Chybí základní prvek vývoje - webové aplikace není funkční.
Podmínky, kdy není nutné pracovní produkt vytvářet	Produkt je nutné vytvořit v každé webové aplikaci.
Způsob reprezentace	Každý ovladač je reprezentován C# třídou. V ní jsou implementovány metody, které se nazývají *akce*. Jejich úkolem je reagovat na URL požadavky, provádět příslušné akce a vracet odpověď uživateli zpět do prohlížeče (Galloway et al., 2014).

5.3.8 Kód webové aplikace

Produkt je definován v metodice MMSP pod názvem Implementace. V rámci rozšíření je přejmenován a přizpůsoben frameworku ASP.NET MVC. Zodpovědnou rolí za produkt je Programátor.

Kód webové aplikace představuje množinu programového kódu, dat a podpůrných souborů (jako např. dokumentace), které slouží k implementaci potřebné funkcionality. V produktu je zahrnut programový kód všech částí aplikace, které pro sebe nemají definován produkt vlastní – není zde tedy zahrnut Pohled, Model a Ovladač. Patří sem především programování klientské části aplikace a konfigurace aplikace včetně směrování.

Klientská část aplikace

K implementaci klientské části aplikace se využívá především JavaScript a knihovny na něm založené, jako je např. nejznámější JQuery. To je k dispozici ihned po vytvoření MVC projektu a jeho výhodou je podpora všemi moderními webovými prohlížeči.

Pro inicializaci skriptů v jednotlivých pohledech je možné využít klasického HTML zápisu, to však přináší nevýhodu v případě změny verze skriptu. K řešení tohoto problému slouží v ASP.NET MVC tzv. Bundler. Ten kromě usnadnění změny verze skriptu přináší výhody např. v automatickém používání minifikované[9] formy či v centralizaci referencí

[9] Minifikací se myslí odstranění nadbytečných dat (prázdné znaky, konce řádků, komentáře atd.) bez pozměnění funkčnosti. Tím jsou ušetřena stahovaná data a je zrychleno načtení stránky.

na skripty, umožňující provádět změny na jednom místě. Bundler lze používat také pro inicializaci kaskádových stylů.

Směrování

Směrování (z angl. routing) ve frameworku ASP.NET zajišťuje zavolání správného Ovladače a jeho akce na základě vstupní URL. Dále také vytváří odchozí URL tak, aby s akcemi ovladačů korespondovaly. Rozdíl oproti tzv. přepisu URL, který mapuje jednu URL na jinou, je ten, že směrování mapuje URL na konkrétní zdroj. Zároveň směrování dokáže reverzně mapovat i odchozí, generované URL.

Směrování je obvykle dobré věnovat pozornost. Lze díky němu zpřehlednit URL pro uživatele, ale zároveň slouží i k optimalizaci pro vyhledávače (SEO).

Tabulka 9: Produkt Kód webové aplikace

Důsledky nezhotovení produktu	Bez tohoto pracovního produktu není možné vytvořit vyvíjenou webovou aplikaci.
Podmínky, kdy není nutné pracovní produkt vytvář**e**t	Implementace by měla být vytvářena vždy.
Zp**ů**sob reprezentace	Implementace má formu souborů uložených v pracovním prostředí.

5.3.9 Unit test

Produkt již v metodice MMSP existuje a zde je popsáno pouze doplnění publikované verze.

„Unit (neboli jednotkové) testy představují specifický typ testů prováděných za účelem ověření funkčnosti malých, soběstačných částí zdrojového kódu implementace“ (MMSP, 2011).

Struktura

U Unit testů je dobré zachovat jednotnou strukturu. Při psaní testů ve frameworku ASP.NET MVC je možné využít struktury zvané *„Arrange, Act, Assert*“ (3A), která vznikla v rámci souboru agilních praktik pro (nejen) extrémní programování (Bill Wake, 2011).

Unit test je tak rozdělen do tří částí:

- Arrange – příprava prostředí (inicializace proměnných, tříd).

- Act – zavolání testovaných metod.
- Assert – ověření, zdali se provedlo to, co mělo.

Tato struktura je společná pro objekty všech typů chování – konstruktory, modifikátory, přístupy (dotazy) a iterátory. Lze ji tak využít v rámci všech unit testů projektu.

5.4 Životní cyklus metodiky

Životní cyklus vývoje softwaru je v metodice MMSP rozdělen do čtyř fází – Zahájení, Rozpracování, Konstrukce a Zavedení. *„V rámci těchto fází může být podle potřeby proveden libovolný počet iterací, které se mohou lišit jak podrobností prováděných úloh, tak svou délkou“* (Buchalcevová a Stanovská, 2013).

Úpravy představené v předchozích kapitolách nemají na strukturu životního cyklu žádný dopad. Doplňují však jednotlivé fáze životního cyklu o nové role, úlohy a produkty. Tabulka 10 zachycuje rozložení realizace těchto nově přidaných úloh v rámci fází životního cyklu metodiky.

Tabulka 10: Realizace úloh v rámci fází životního cyklu

Disciplína / Úloha	Fáze			
	Zahájení	Rozpracování	Konstrukce	Zavedení
Architektura				
Tvorba wireframe				
Vývoj				
Návrh grafického designu				
Převod designu do pohledů				
Návrh databáze				
Implementace webové aplikace				
Testování				
Příprava testů				
Tvorba unit testů				

Důležitou úlohou, na kterou následně navazuje většina úloh z disciplíny Vývoj je Tvorba wireframe, která je realizována od fáze Zahájení do fáze Konstrukce. Úlohy z disciplíny Vývoj jsou realizovány především ve fázi konstrukce. Úlohy disciplíny Testování naopak pokrývají celý životní cyklus projektu, od specifikace předmětu a způsobu testování, strategie a způsobu reportování chyb až po samotnou realizaci testů.

V závislosti na projektu se mohou lišit počty iterací v jednotlivých fázích. Rozdělení definované v předchozí tabulce by však mělo být v rámci naprosté většiny projektů neměnné, neboť vývoj probíhá nad stejnou architekturou.

5.5 Návody

V této kapitole je popsána úprava posledního stavebního prvku metodiky nazývaného Návod, nebo také Řízení (Guidance), který obsahuje několik dalších volitelných prvků. „Prvky jsou spojeny s úkoly, pracovními produkty nebo prvky procesů. Počet Řízení, které se mohou využít k jednotlivým prvkům, není omezen. Prvky Řízení mohou být propojovány“ (Šimko, 2015).

Tabulka 11: Kategorie oblasti Řízení

Název	Obsah
Kontrolní seznamy (checklists)	Obsahují seznam položek, které musejí být dokončeny nebo ověřeny.
Koncepty (concepts)	Obsahují klíčové principy, základní myšlenky a doplňující informace k danému produktu či úloze. Zabývají se obvykle obecnějšími tématy než směrnice.
Příklady (examples)	Představují typické, částečně vyplněné, instance nějakého prvku metodiky. Nejčastěji jsou použity pro produkty.
Směrnice (guidelines)	Poskytují informace o tom, jak zacházet s konkrétními prvky obsahu metodiky. Nejčastěji se vztahují k produktům či úlohám.
Praktiky (practices)	Obecně osvědčené postupy užívané v rámci realizace projektu.

Mnoho prvků patřících do jedné z kategorií oblasti Řízení, představených v předchozí tabulce, je z důvodu ucelenosti textu definováno v jiných kapitolách. V publikované verzi upravené metodiky v nástroji EPFC jsou samozřejmě prvky zaznamenány samostatně a přiřazeny pomocí referencí k příslušným úlohám či produktům.

V následujících podkapitolách jsou popsány prvky z oblasti Řízení, které nebyly definovány v jiných kapitolách práce, ať už z důvodu rozsahu, zachování přehledné struktury práce, či skutečnosti, že se váží k několika prvkům metodiky zároveň.

5.5.1 Konvence při vývoji v ASP.NET MVC

Framework ASP.NET MVC zastává koncept „Convention over Configuration“, tedy dává přednost dodržování konvencí před udržováním mnoha konfiguračních souborů. Dodržením dané struktury a názvových konvencí se programátor zbavuje nutnosti konfigurace a explicitního nastavování a může se tak plně soustředit na vývoj aplikace.

Po vytvoření projektu ve vývojovém prostředí Visual Studio dle jedné z předdefinovaných šablon se zároveň zakládá základní struktura složek a souborů (Tabulka 12). Struktura neslouží pouze k zajištění přehlednosti v projektu, vyplývá totiž z definovaných pravidel. Každý vývojář by měl tato pravidla dobře pochopit a dodržovat pro pohodlí vlastní implementace a snazší sdílení kódu mezi více vývojáři, ale i pro pochopení vlastního kódu v budoucnu (Galloway et al., 2014).

Tabulka 12: Základní adresáře a jejich význam v rámci projektu

Adresář	Účel
/App_Data	Datové soubory, jako jsou XML soubory či souborově založené databáze.
/App_Start	Konfigurační soubory pro směrování, bundling atd.
/Content	Statický obsah jako jsou CSS, obrázky a vše ostatní (kromě skriptů).
/Controllers	Třídy reprezentující Ovladače, spravující URL požadavky.
/fonts	Fonty využívané v aplikaci.
/Models	Třídy (Modely), které reprezentují a manipulují s daty a business objekty.
/Scripts	JavaScript knihovny a skripty (.js), jQuery a AJAX knihovny jsou v této složce již ve výchozím projektu.
/Views	Pohledy neboli UI šablony, zodpovědné za renderování výstupu. Obvykle jsou umisťovány dále do podsložek, které odpovídají příslušným ovladačům.
/Views/Shared	Layout a pohledy, které nepatří jen jednomu Ovladači.

Kromě zmíněné adresářové struktury existuje v projektu také několik klíčových konfiguračních souborů, vypsaných v následující tabulce. Zvláštností je existence dvou stejně pojmenovaných konfiguračních souborů Web.config, tím lze zajistit specifickou konfiguraci v rámci adresářové hierarchie pro specifické třídy aplikace (MSDN, 2016).

Tabulka 13: Základní soubory a jejich význam v rámci projektu

Soubor	**Úč**el
/Web.config	Hlavní konfigurační soubor aplikace.
/Views/Web.config	Konfigurace pro správně fungování Pohledů a zamezení jejich přímého přístupu na serveru.
/Global.asax	Globální třída využívaná k registraci filtrů či směrování při startu aplikace.

Aby automatická konfigurace fungovala, je třeba kromě zachování správně struktury v rámci projektu používat i správná pojmenování:

- název ovladače musí končit řetězcem `Controller` (např. `AccountController`),
- pohledy by měly být vytvářeny do složky `/Views/NázevOvladače` nebo `/Views/Shared`,
- výchozí pohled pro akce by měl být pojmenovaný stejně jako příslušná metoda (např. akci `Index` by měl náležet pohled `Index.cshtml`),
- layout by měl být pojmenovat s podtržítkem na začátku názvu (např. `_Layout.cshtml` a měl by být umístěn ve složce `/Views/Shared`,
- vstupní pole v HTML formuláři by měla mít stejný název jako atribut modelu, kterému odpovídá – může být zajištěno použitím HTML helpers.

Jak již bylo řečeno, dodržení zmíněných konvencí minimalizuje množství nutných konfigurací k běhu aplikace. Není např. nutné specifikovat, který k jakému ovladači patří daný pohled, či kde se nachází. Většinu zmíněných konvencí lze ve výjimečných případech porušit a vše ručně specifikovat v souboru Web.config. Důvodem může být např. rozdělení velké aplikace do několika lépe spravovatelných celků. V naprosté většině aplikací je však silně doporučeno se konvencemi řídit.

5.5.2 Osvědčené postupy při vývoji v ASP.NET MVC

Osvědčené postupy neboli best practices jsou metody či techniky, které jsou obecně přijímány jako nejvýhodnější. Vznikají postupem času na základě kladných referencí od vývojářů a představují tak jakési šablony či standarty, jejichž využití je doporučeno v případě vývoje každé aplikace. Následující výpis obsahuje základní postupy využívané při vývoji ve frameworku ASP.NET MVC, které jsou obvykle publikovány v několika zdrojích a využívány vývojáři po celém světě.

Model

1. Vkládat veškerou business logiku do modelů (MSDN, 2016).

   Je doporučeno odstínit pohledy a ovladače od rozhodnutí ovlivňující data. Výhodou je méně duplicit business logiky, větší přehlednost pohledů oproštěných od této logiky a jednodušší testování pravidel izolovaných v modelu.

2. Vkládat veškerou validační logiku do modelů.

Pohled

1. HTML kód vkládat pouze do pohledů.
2. Využívat parciálních pohledů (TechFunda™, 2016).

   Parciální pohled obsahuje nějakou část stránky, která se vloží do pohledu hlavního. Využívá se, pokud je stejný kus kódu použit ve více pohledech, nebo pro modální okna.

3. Přistupovat k datům pouze pomocí ViewData, ViewBag či modelu pohledu (MSDN, 2016).
4. Využívat automatické klientské validace na základě modelu.
5. Využívat komentáře na straně serveru (MSDN, 2016).

   Nepoužívat klasické HTML komentáře, které jsou předávány klientovi a je možné je číst ve zdrojovém kódu stránky. Komentáře na straně serveru se použijí následujícím způsobem: `<%- text komentáře -%>`.

6. Využívat HTML helpery.
7. Nenechat skripty zpomalit načítání stránky (Galloway et al., 2014).

   Skripty načítat vždy až po načtení daného pohledu, k tomu lze využít speciální tag `@section Scripts`. Do layoutu stránek vkládat pouze skripty, které jsou využívané na všech stránkách.

Ovladač

1. Využívat vazbu s modelem (MSDN, 2016).

   Pro získávání dat z pohledu, například při odeslání formuláře, využívat tzv. model binding. Jedná se o mechanizmus, který pomocí reflexe převede kontextová data požadavku do definovaných objektů. Tímto způsobem lze i snadno validovat vstupní data.

2. Ošetřit standartní chyby.

Změnit výchozí reakce na chybové akce. Např. místo zobrazení chyby 404 (nenalezeno) je možné přetížením metody `HandleUnknownAction` uživateli zobrazit vlastní stránku.

Wireframe

1. Začít zeširoka a experimentovat.

   Tvorba wireframů je rychlá a nepřináší téměř žádná rizika (Garret, 2011). Z tohoto důvodu je možné při návrhu více experimentovat a docílit tím lepších výsledků.

2. Soustředit se na strukturu, nikoli na detail.

   Detail bude zpracováván v dalších krocích vývoje. Použití barev či obrázků při návrhu wireframe se nedoporučuje, vede to pouze k nerelevantní zpětné vazbě.

3. Jednoduchost.

   Je třeba si uvědomit, že každý element wireframu bude muset být posléze programován. Je tedy vhodné vyhnout se složitým návrhům.

4. Použití reálných textů.

   Doporučuje se udávat texty, jako jsou názvy tlačítek, záložek, nadpisů, ve finálním znění. Při nedodržení tohoto bodu hrozí nutnost změny velikostí jednotlivých elementu apod.

5. Zachování konzistence.

   V případě vytváření více wireframů je důležité zachovat stejné parametry pro jednotlivé elementy, které se opakují.

6. Zaměření na obsah.

   Navrhovat v závislosti na cílovou skupinu uživatelů a zaměření webu. V ideálním případě uživatel přestane samotné prostředí aplikace po době užívání vnímat a bude se soustředit na obsah. Wireframe by měl zvýrazňovat důležitý obsah, jako je např. tlačítko zakoupení produktu atd. (Bank a Waleed, 2014).

Obecné

1. Udržovat čistou URL adresu (Freeman, 2013)

   Pro uživatele je mnohem příjemnější, když je zobrazovaná URL adresa srozumitelná. Navíc takovou adresu umí lépe zpracovávat vyhledávače. Je tedy doporučeno snažit se vyvarovat následujícímu formátu adresy:

```
http://www.amazon.com/Pro-ASP-NET-MVC-Professional-Apress/dp/
1430242361/ref=la_B001IU0SNK_1_5?ie=UTF8&qid=1349978167&sr=1-5
```

Místo toho využívat čistou a srozumitelnou adresu:

```
http://www.amazon.com/books/pro-aspnet-mvc5-framework
```

2. Veškeré názvy (proměnných, tříd, anotací atd.) psát v anglickém jazyce.
3. Využívat filtry.

   Filtry ve frameworku ASP.NET MVC umožňují spuštění nějaké logiky před nebo po vykonání akce v ovladači. Je to velmi mocný nástroj, který lze využít např. pro autorizaci, logování či ukládání dat do cache.
4. Psát unit testy

   Architektura frameworku zajišťuje vysokou míru testovatelnosti jednotlivých částí – je jednoduché oddělit a testovat např. business logiku uvnitř modelů apod. Je doporučeno unit testů při vývoji aplikace vždy využívat.
5. Využívat autentizace a autorizace uživatelů pro ochranu obsahu.
6. Využívat k obnovení jednotlivých částí stránek AJAX, což šetří přenesená data a zrychluje zobrazení.
7. Využívat cache pro dočasné uložení statických stránek.

   Je vhodné uložit statický obsah, jako je např. úvodní stránka do mezipaměti. Stránka pak nebude při každém požadavku načítána znova a její obnovení bude probíhat v definovaném intervalu.
8. Využívat asynchronní ovladače pro déle trvající požadavky.
9. Minifikace a bundling CSS a skriptů (Freeman, 2013)

   Cílem minifikace je snížit velikost stahovaných dat při načítání stránky. Minifikací se rozumí odstranění z daných souborů prázdných mezer. V případě skriptů se provádí operací více – např. jsou přejmenovány proměnné na jednoznakové názvy. Samotný proces nemá žádný vliv na funkčnost a k realizaci se využívají již připravené nástroje.

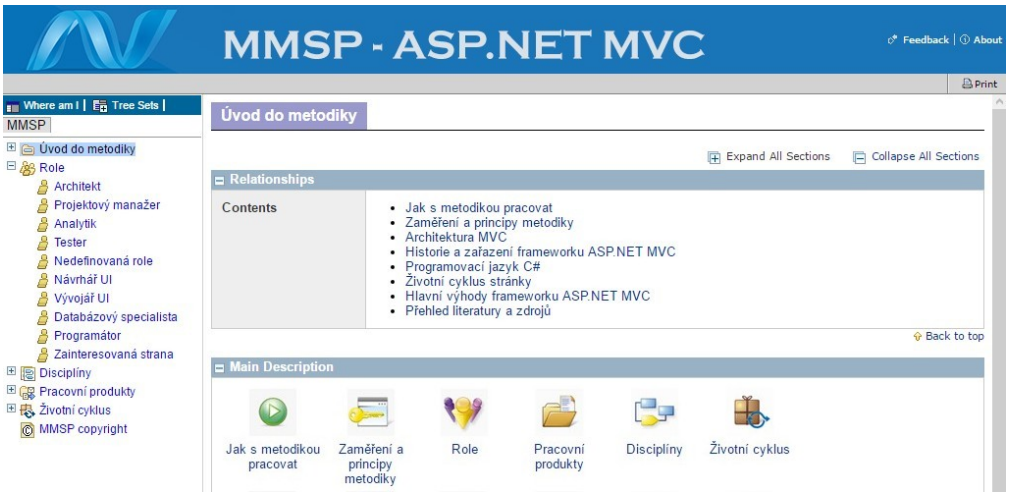
   Bundling poté snižuje počet požadavků na daný server tím, že jednotlivé skripty (nebo CSS soubory) spojí do jednoho, jehož stažení je poté rychlejší.

5.6 Publikace metodiky

Metodika MMSP – ASP.NET MVC, jejíž návrh byl představen v předchozích kapitolách, byla publikována pomocí nástroje Eclipse Process Framework Composer (EPFC) a je zveřejněna na stránkách https://kitscm.vse.cz. *Prostřednictvím publikované webové verze je možné na metodiku nahlížet z různých úhlů pohledu a bezproblémově přecházet mezi souvisejícími rolemi, úlohami a pracovními produkty* (Rejnková, 2011).

„EPFC je open-source nástroj pro správu metodiky a je dostupný zdarma. Nástroj se využívá k udržování projektových postupů, procesů, správu metodik, norem a standardů. Je určen metodikům, procesním inženýrům, vedoucím projektů, projektovým manažerům a programovým manažerům, kteří zodpovídají za realizaci, údržbu a rozvoj procesů v rámci organizace nebo projektu“ (Šimko, 2015).

Jako základ pro publikaci nové metodiky byla použita výchozí metodika MMSP, která je v nástroji EPFC publikována také. Ovládání nástroje není úplně triviální, pro implementaci potřebných úprav a naučení se s nástrojem jsem tak využil několika zdrojů. Hlavním z nich byla diplomová práce (Rejnková, 2011), v rámci které vznikla metodika MMSP. Autorka zde poměrně podrobně popsala celý postup tvorby metodiky a především zvolený systém překrývání prvků a lokalizace, kterým nahradila výchozí metodiku OpenUP. Obecnější pokyny, jak s nástrojem pracovat byly získány především z diplomové práce popisující nástroj EPFC (Šimko, 2015). Dalším zdrojem, především pro technické zpracování, byly webové stránky nástroje (EPFP, 2016).



Obrázek 9: Úvodní stránka publikované metodiky MMSP-ASP.NET MVC

Metodika MMSP byla vytvořena jako plug-in k metodice OpenUP, který některé její části nahrazuje a jiné přidává. Výhodou je zachování konceptu metodiky s veškerými vztahy mezi jednotlivými prvky. Při vytváření metodiky MMSP – ASP.NET MVC jsem využil tohoto plug-inu a rozšířil ho o nové Role, Úlohy a Pracovní produkty, definované v předchozích kapitolách. Ukázky publikované metodiky jsou k dispozici v Příloze A.

6 Ověření metodiky MMSP – ASP.NET MVC

V této kapitole je popsán proces vývoje aplikace pro správu databáze antropologického oddělení Národního muzea za využití metodiky MMSP – ASP.NET MVC. Touto částí práce jsem přímo navázal na svou práci bakalářskou (Velemínský, 2014), ve které jsem databázi pro AONM navrhoval. Pro lepší pochopení celého procesu nejprve stručně popíši, jaká data jsou v databázi uložena, co je hlavním cílem aplikace, jak a kým bude využívána. Následně se zaměřím na samotný proces jejího vývoje za využití metodiky MMSP ASP.NET MVC. Aktuální verzi aplikace je možné nalézt na anthroponm.tk. Pro přihlášení na uživatele je možné využít přihlašovací jméno *guest* s heslem *password*. Vzhledem k tomu, že jsou v databázi již nahrána aktuální data antropologického oddělení, má tento uživatel oprávnění pouze na čtení.

6.1 O antropologickém oddělení Národního muzea

Antropologické oddělení patří k nejmladším oddělením přírodovědné části Národního muzea. Bylo založeno v roce 1967 profesorem Emanuelem Vlčkem. Sbírky kosterních nálezů však byly získávány již od počátku existence Národního muzea, tedy od počátku 19. století. V současné době na tomto oddělení pracuje devět zaměstnanců.

Práce antropologa v muzeu představuje především tvorbu a vědecké zpracování sbírek, s následnou popularizací výzkumů formou expozice, výstav či článků. Z důvodu obtížnosti získávání materiálu vlastním výzkumem, většina muzejních antropologických pracovišť, včetně toho v Národním muzeu, úzce spolupracuje s archeology (Dobisíková a Anděra, 1999).

Od svého založení 1. října 1967 oddělení shromáždilo více než 25 tisíc kosterních nálezů, které získalo převážně z archeologických výzkumů na území bývalé ČSSR, později České republiky. Vytvořilo tak jednu z největších osteologických sbírek v Evropě, která je stále rozšiřována. Pro sbírky byl v roce 2001 vybudován nový moderní depozitář, čímž byl dán předpoklad k soustředění nálezů do jednoho místa a k možnosti jejich soustavného vědeckého zpracování v širokém kontextu.

Hlavním obsahem databáze je tedy právě antropologická podsbírka Národního muzea, která je již od svého vzniku rozdělena do čtyř základních sbírkových celků. V rámci každého celku jsou uchovávány jiné údaje či s nimi pracuji jiní zaměstnanci.

Vyvíjená aplikace bude sloužit ke každodennímu využití zaměstnanci antropologického oddělení. Hlavním požadavkem je dobrá dostupnost, což je zajištěno právě formou webové aplikace. Aplikace musí být schopna evidovat všechny požadované informace o

daných sbírkách a musí podporovat přihlášení z více účtů s různě definovanými právy. Jednotlivé záznamy musí jít v rámci aplikace přidávat, editovat i mazat, přidávat k nim soubory, jako jsou fotografie či jiné dokumenty a to vše musí být podpořeno dobrou použitelností. Kromě evidence sbírkových předmětů je dále od aplikace požadována evidence výpůjční smluv a publikací, které jsou se sbírkovými předměty spjaty.

6.2 Využití metodiky MMSP – ASP.NET MVC

V průběhu vývoje aplikace byla využita metodiky MMSP–ASP.NET MVC. Na vývoji aplikace jsem pracoval sám a je tak zřejmé, že nebyl využit celý potenciál metodiky ve smyslu týmové koordinace a spolupráce.

V této kapitole jsou uvedeny postupy a produkty, které jsem v rámci zpracování jednotlivých Úloh vytvořil. Jsou zde uvedeny poznámky k jejich tvorbě a reference na obsah metodiky MMSP-ASP.NET MVC. Jedná se především o prvky metodiky z disciplíny Vývoj, které jsem se v rámci rozšíření věnoval.

6.2.1 Iterativní vývoj

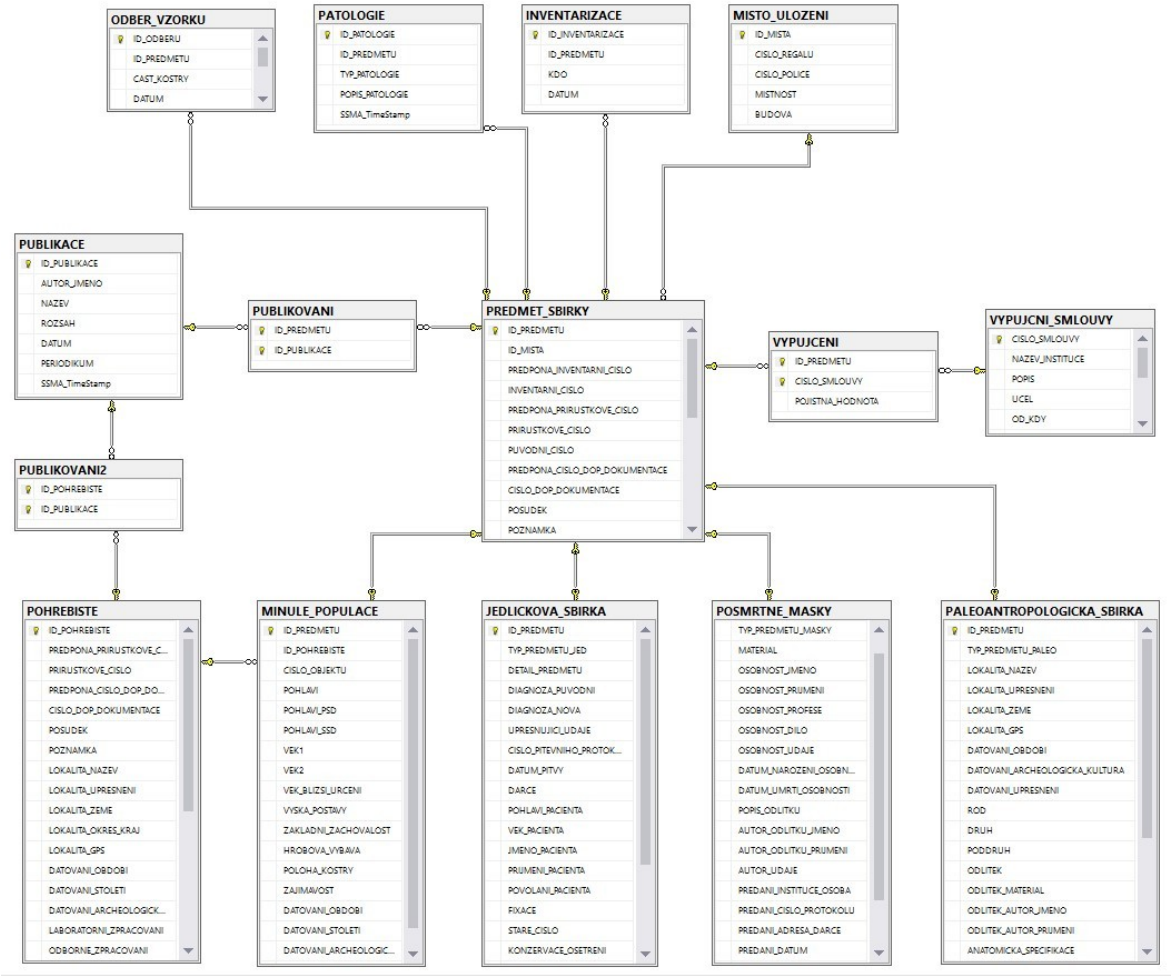
V průběhu celého vývoje jsem dodržoval iterativní životní cyklus metodiky. Produkt Build webové aplikace, byl vytvářen v co nejkratších intervalech, díky čemuž se navzájem propojené prvky aplikace musely vytvářet postupně a snadněji se následně ladily chyby. Další výhodou byla častá zpětná vazba od zadavatele, snadnější změny a agilní průběh vývoje.

6.2.2 Datový model

Při implementaci aplikace jsem zvolil přístup *database first*[10]. Důvodem byl především fakt, že účelem aplikace je právě správa databáze, což činí datový model nejkritičtějším produktem při vývoji. Datový model byl navrhován důkladně a před jeho finální podobou prošel mnoha změnami. Díky tomu však při implementaci samotné aplikace nebylo datový model téměř třeba upravovat. Na následujícím obrázku je zachycen navržený fyzický model (z důvodu čitelnosti nejsou u tabulek zobrazeny všechny atributy a datové typy jsou skryty).

[10] Database First je další z přístupů inicializace databáze pomocí objektově-relačního mapovače zvaného Entity Framework. Na rozdíl od přístupu Code First vyžaduje předem vytvořenou databázi, na základě které následně definuje objekty pro práci s daty.

Jádrem modelu je tabulka *předmět sbírky*, která obsahuje společné atributy pro všechny sbírky. Tabulky jednotlivých sbírek jsou na ní následně navázány vztahem 1:1. Zároveň slouží jako propojení s ostatními tabulkami jako *Inventarizace, uložení, výpůjční smlouvy* atd., což několikanásobně snižuje počet vazeb modelu.

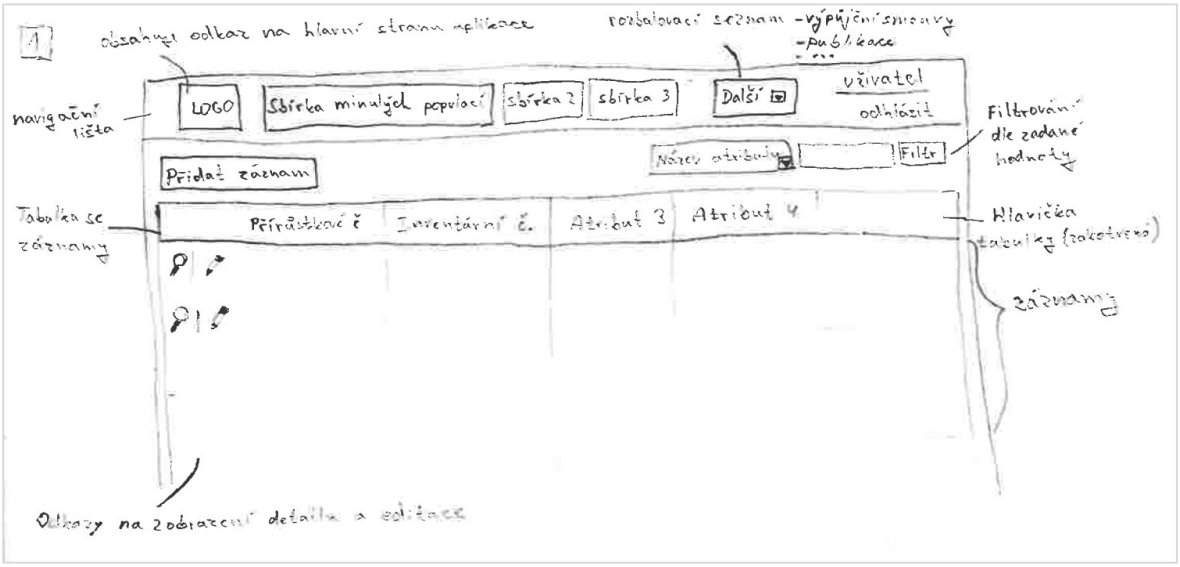


Obrázek 10: Fyzický databázový model

6.2.3 Wireframe

Vzhledem k tomu, že vyvíjená aplikace obsahuje pouze malý počet typů obrazovek, využil jsem k tvorbě wireframu náčrt tužkou. První takto zpracovaný wireframe, který zobrazuje obrazovku s výpisem předmětů pro zvolenou sbírku, je zachycen na následujícím obrázku.

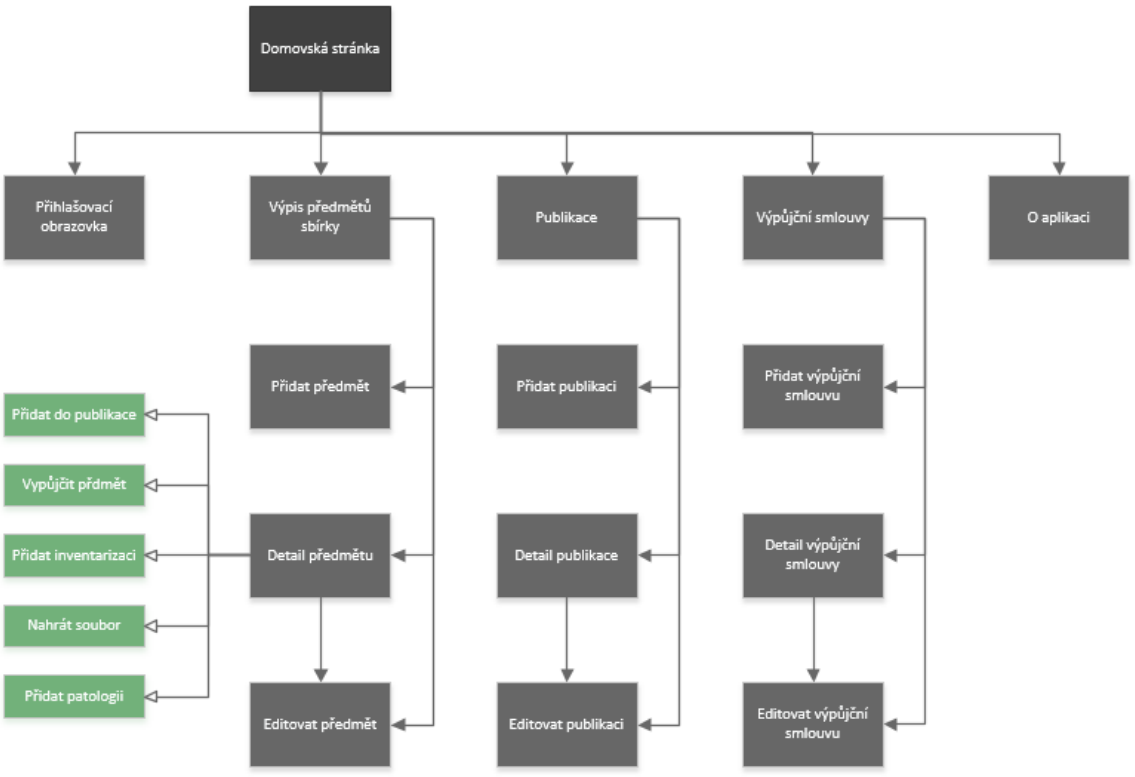
Součástí každé obrazovky aplikace je navigační lišta, obsahující logo s odkazem na úvodní stránku aplikace a odkazy na jednotlivé sbírky. Kromě odkazů je v pravé části navigační lišty zobrazen také odkaz na přihlášení / odhlášení uživatele a případně jméno aktuálně přihlášeného.



Obrázek 11: Wireframe – výpis předmětů sbírky

Při vytváření produktu byla dodržena pravidla definovaná v metodice. Kde je to možné, uvádím finální názvy elementů a jednotlivé elementy jsou dále specifikovány. Ve wireframu je zachycen design informační, navigační i design UI.

6.2.4 Navigační mapa

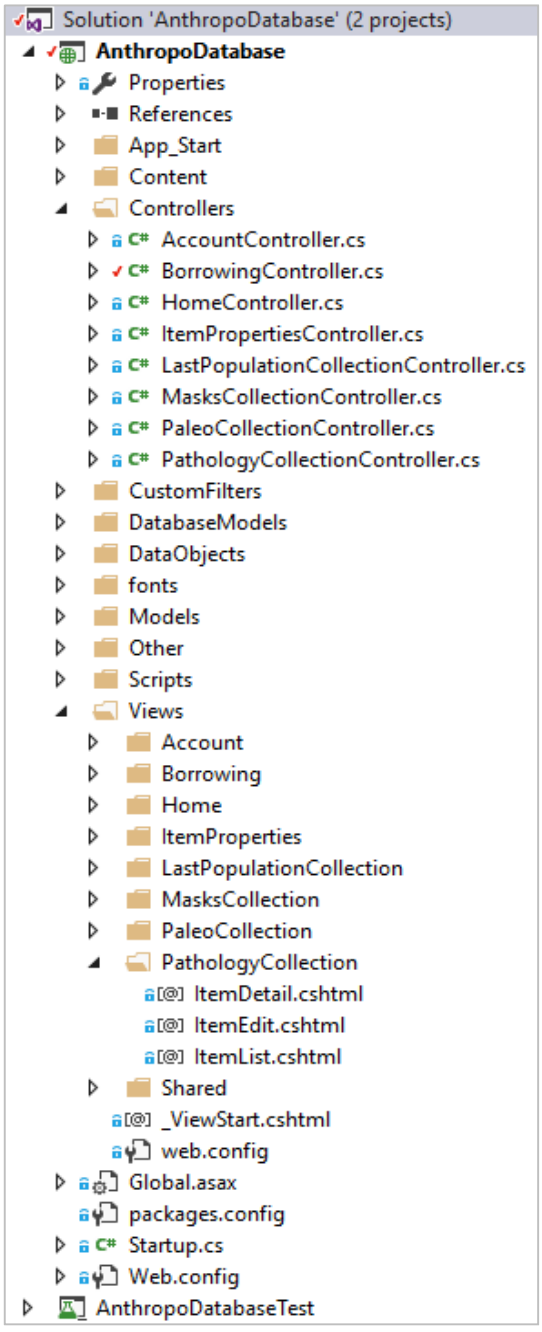


Obrázek 12: Navigační mapa

Navigační mapa je zpracována dle pravidel definovaných v rámci metodiky MMSP-ASP.NET MVC. Pro tvorbu jsem využil grafu volného, neboť jsem chtěl zaznamenat více vazeb, než by stromový graf povolil. Zeleně jsou v diagramu označeny stránky zobrazované v dialogovém okně.

6.2.5 Struktura projektu a konvence

Obrázek 13 zachycuje struktura projektu, vycházející z konvencí definovaných v rámci předchozí kapitoly.



Obrázek 13: Struktura projektu

Jednotlivé třídy a části aplikace jsou umístěné v odpovídajících adresářích. Na obrázku je možné vidět i využití jmenných konvencí, kdy názvy složek s pohledy odpovídají názvům příslušných ovladačů. Zároveň poté názvy jednotlivých pohledů odpovídají názvům akcí v daném ovladači.

V projektu je vytvořeno i několik dalších složek, nevycházejících přímo z konvencí:

- CustomFilters - obsahuje implementace vlastních filtrů využitých v aplikaci. Jeden z těchto filtrů je popsán v následující kapitole,
- DatabaseModels – obsahuje databázové schéma vygenerované na základě již vytvořené databáze pomocí Entity Frameworku,
- DataObjects – implementace rozhraní pro přímou práci s databází pro jednotlivé datové objekty. Tyto třídy jsou volány pouze z modelů,
- Other – obsahuje pomocné třídy využívané na více místech aplikace. Jedná se např. o HTML helpery, implementace autentizace či pomocné metody.

6.2.6 Vlastní implementace Filtru

Filtru bylo v aplikaci využito k implementaci autorizace. V rámci aplikace je možné vytvářet uživatelské účty s různými úrovněmi oprávnění. Každý uživatel má právo na čtení celé databáze, práva na editaci jednotlivých sbírek lze však jednotlivě u každého účtu nastavit.

Filtr se pak používá k ověření, zdali má aktuálně přihlášený uživatel právo na spuštění dané funkčnosti. Toto ověření probíhá díky vlastnostem filtru před samotným spuštěním akce v příslušném ovladači a v případě, že uživatel práva nemá, je zobrazena odpovídající chybová zpráva.

6.2.7 Vlastní HTML helper

Jak bylo definováno v rámci metodiky ASP.NET MVC, HTML helper slouží ke generování HTML kódu, kdy je pomocí C# kódu možné přidání nějaké logiky. Kromě předdefinovaných HTML helpers jsem zmínil možnost vytvoření vlastního.

Na následujícím výpisu je zachycena zjednodušená implementace helperu vytvořeného v rámci vyvíjené aplikace. Zajišťuje funkčnost zvýraznění tlačítka aktuálně prohlížené sbírky / sekce webu v navigační liště. Realizace je uskutečněna pomocí vytváření odkazu, který má přidělenou konkrétní CSS třídu, pokud se Ovladač z odkazu tlačítka shoduje s Ovladačem aktuálně zobrazené stránky. Jak je možné na přikladu vidět, tvorba odkazu je realizována opět pomocí HTML helperu, tentokrát již předdefinovaného samotným frameworkem.

Výpis 3: Implementace vlastního HTML helperu

```
public static MvcHtmlString MenuLink(this HtmlHelper htmlHelper,
                                      string linkText,
                                      string actionName,
                                      string controllerName)
{
   string currentController = htmlHelper.ViewContext.RouteData.GetRequiredString("controller");

   if (controllerName == currentController)
   {
            return htmlHelper.ActionLink(linkText, actionName, controllerName, null,
                     new { @class = "selected" });
   }

   return htmlHelper.ActionLink(linkText, actionName, controllerName);
}
```

6.2.8 Model a validace

Podstatnou část každé webové aplikace jsou validace. Dle vytvořené metodiky je doporučeno validace zpracovávat společně s modelem, který jejich implementaci usnadňuje.

Na následujícím výpisu je možné vidět validaci dvou atributů typu datum pomocí anotací, která se následně v aplikaci projeví jak na straně klienta (díky integrované knihovně jQuery), tak serveru.

Kromě validace pomocí předdefinovaných anotací je implementována také validace vlastní, která u výpůjček zajišťuje, aby datum do bylo pozdější než datum od. Toho je dosaženo tak, že třída `BorrowingModel` implementuje rozhraní `IValidatableObject` a potřebné chování validace je následně ošetřeno v metodě `Validate`. Takto vytvořená validace se následně projeví jen na straně serveru. To v praxi znamená, že se formulář po stisknutí tlačítka na odeslání neodešle, ale vrátí se s odpovídající chybou.

Výpis 4: Využití validace pomocí anotací a vytvoření vlastní validace

```
public class BorrowingModel : IValidatableObject
{
    [Display(Name = "Od kdy")]
    [DataType(DataType.Date)]
    [DisplayFormat(DataFormatString = "{0:dd. MM. yyyy}")]
    [Required(ErrorMessage = "Atribut {0} je povinný")]
    public DateTime DateFrom { get; set; }

    [Display(Name = "Do kdy")]
    [DataType(DataType.Date)]
    [DisplayFormat(DataFormatString = "{0:dd. MM. yyyy}")]
    [Required(ErrorMessage = "Atribut {0} je povinný")]
    public DateTime DateTo { get; set; }

    public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)
    {
        if (DateTo < DateFrom)
        {
            yield return
                  new ValidationResult(errorMessage: "Datum do musí být pozdější než datum od",
                                       memberNames: new[] { "DateTo" });
        }
    }
}
```

6.2.9 Unit test

Testování by mělo být nedílnou součástí každého projektu a ani v rámci vývoje aplikace pro AONM jsem ho nevynechal. V průběhu implementace jsem využíval k otestování

jednotlivých funkčností unit testy. Ty se v ASP.NET MVC vytvářejí v odděleném projektu, asociovanému k projektu hlavnímu a správné provádění testů se zajišťuje anotacemi nad třídami a metodami (viz. anotace TestClass a TestMethod v následujícím výpisu).

Ukázkový test dodržuje strukturu 3A, kterou jsem definoval v rámci kapitoly 5.3.9. Jedná se o jednoduchý test, který kontroluje správné chování ovladače pro sbírku patologických změn, konkrétně pak akci pro přidání nového předmětu či úpravu stávajícího. Pokud se akce zavolá bez parametrů, má se vrátit pohled a jeho atributy s odpovídajícími hodnotami pro přidání nového předmětu.

Výpis 5: Ukázka jednoduchého unit testu se strukturou 3A

```
[TestClass]
public class PathologyCollectionControllerTest
{
   [TestMethod]
   public void NewItemTest()
   {
        // Arrange
        PathologyCollectionController controller = new PathologyCollectionController();

        // Act
         var result = controller.ItemEdit(null, null, null) as ViewResult;

        // Assert
        Assert.IsNotNull(result);
        Assert.AreEqual(true, result.ViewData["NewItem"]);
   }
}
```

6.2.10 Využití knihoven

Tak jak je v prvním kroku úlohy Implementace webové aplikace definováno, je dobré před samotnou tvorbou netriviálních elementů webových stránek rozhodnout, jakým způsobem bude implementace probíhat a zjistit, zdali již neexistuje nějaká knihovna či zásuvný modul, které by potřebnou funkci zastávaly. Pro realizaci následujících funkčností jsem se rozhodl využít již hotových řešení:

- dialog s výběrem data,
- fixní hlavička tabulky při posunu stránky níže,
- modální okna.

6.3 Shrnutí

V rámci procesu ověření jsem prošel všemi fázemi metodiky MMSP-ASP.NET MVC od zahájení, kde probíhala především analýza a upřesňování požadavků se zadavateli, po fázi nasazení, kdy byl spuštěn provoz hotové webové aplikace. Snímky vytvořené aplikace je možné nalézt v příloze B.

Celkově projekt trval přibližně 6 měsíců. Vzhledem k tomu, že jsem využil definice požadavků a návrhu datového modelu ze své bakalářské práce, úlohy analýzy byly podstatně kratší, než v případě, kdy bych se s danou problematikou setkal poprvé. Ačkoliv jsem měl poměrně rozsáhlý základ, bylo třeba se zadavateli diskutovat a sestavit požadavky na samotnou aplikaci a její funkčnosti. Vývoj aplikace poté probíhal iterativně, po menších, ucelených částech konzultovaných se zadavateli tak, aby bylo předejito případným nedorozuměním a změnám. V každé iteraci byly vykonávány úlohy definované metodikou v pořadí, které bylo zrovna nasnadě, neboť pořadí úloh v metodice není pevně určeno. To mělo na následek flexibilní vývoj a možnost soustředění se na hlavní cíle dané iterace. Jednotlivé produkty, jakožto výstupy prováděných úloh, byly popsány v předchozí kapitole společně s popisem jejich tvorby.

Webová aplikace byla nasazena do provozu v průběhu září. Od té doby bylo implementováno několik změn, plynoucích z použití v reálném provozu. Jednalo se převážně o změny drobné a snadno zvládnutelné, čehož bylo dosaženo především důslednou analýzou a návrhem. Díky využití metodiky MMSP-ASP.NET MVC nebyla opomenuta žádná důležitá stránka vývoje a byla nasazena funkční, použitelná aplikace, která je využívána zaměstnanci antropologického oddělení při jejich každodenní práci.

7 Závěr

Hlavním cílem diplomové práce bylo rozšíření Metodiky pro Malé Softwarové Projekty (MMSP) pro projekty zabývající se vývojem webových aplikací ve frameworku ASP.NET MVC. Hlavní cíl byl splněn na základě postupného splnění předem vytyčených dílčích cílů.

Za tímto účelem jsem v třetí kapitole charakterizoval specifika vývoje webových aplikací, která mají řadu odlišností oproti běžným aplikacím. Tyto odlišnosti vyplývají především z možností technologií, na kterých jsou založené. Webové aplikace se odlišují také způsobem použití, z čehož plyne nutný důraz na bezpečnost, rychlost a použitelnost. Tyto aspekty bylo nutné v následné úpravě metodiky MMSP zohlednit.

V další části třetí kapitoly jsem se zabýval agilním vývojem a jeho využitelností v rámci vývoje webových aplikací. Vzhledem k mnoha aspektům, jako je rychlost vývoje, iterativnost, snadnější přijímání změn či práce v menších týmech, jsem agilní přístup shledal pro webové aplikace vhodným a zaměřil jsem se na samotnou metodiku MMSP. Popsal jsem základní prvky a charakteristiky metodiky a především analyzoval, zdali je právě tato metodika vhodná pro plánované úpravy.

Abych byl schopen navrhnout úpravu metodiky MMSP, zabýval jsem se ve čtvrté kapitole práce podrobněji frameworkem ASP.NET MVC. Charakterizoval jsem platformu ASP.NET, na které je framework postaven a krátce popsal historii jeho vývoje a důvody k jeho vzniku. Popsal jsem základní specifika jazyka C#, vývojové prostředí a princip architektury MVC, která je podstatná pro pochopení celého frameworku a především pro následný správný návrh aplikací. V poslední části kapitoly jsem shrnul hlavní výhody zvoleného frameworku, které ho činí konkurenceschopným v oblasti webového vývoje. Zaměřil jsem se na jeho architekturu, rozšiřitelnost, testovatelnost, či možnosti směrování. Obsah této kapitoly byl zakomponován do upravené metodiky a to především ve formě směrnic a konceptů, které patří do oblasti metodiky zvané Návody.

Ve stěžejní, páté kapitole této práce jsem se věnoval návrhu samotné metodiky MMSP-ASP.NET MVC. Návrh jsem rozdělil do několika fází, odpovídajícím oblastem metodiky MMSP. Nejprve jsem se zaměřil na Role, kde jsem obecnou roli Vývojář rozdělil do čtyř specializovanějších rolí – Návrhář UI, Vývojář UI, Databázový specialista a Programátor. U každé role jsem definoval její účel, zodpovědnosti, požadované dovednosti a možnosti jejího obsazení. Následně jsem se věnoval tvorbě Úloh, které jsou rolemi (převážně nově vytvořenými) vykonávány. Celkem jsem navrhl, případně doplnil, šest Úloh – Tvorba wireframu, Návrh grafického designu, Převod designu do pohledů, Návrh databáze, Implementace webové aplikace a Tvorba unit testů. U každé Úlohy byl uveden její popis a účel, zařazení v rámci disciplín metodiky, vstupy a výstupy, role

vykonávající úlohu a potřebné kroky k jejímu vykonání. Zmíněné Úlohy byly zároveň doplněny do jednotlivých fází Životního cyklu metodiky, díky čemuž jsou zahrnuty v procesu vývoje webové aplikace. V rámci další fáze úpravy metodiky jsem se zabýval Pracovními produkty, které vznikají jako výsledky jednotlivých Úloh. Při úpravě produktů metodiky MMSP došlo kromě vzniku nových i k úpravě stávajících. Nových či upravených Pracovních produktů je celkem devět – Wireframe, Navigační mapa, Grafický design webové aplikace, Pohled, Datový model, Model, Ovladač, Kód webové aplikace a Unit test. U každého produktu byl uveden účel, zodpovědná role, důsledky nezhotovení produktu, forma, ve které by měl být vytvořen a případy, kdy produkt není třeba vytvářet. Poslední upravovanou oblastí metodiky byly tzv. Návody. Jedná se o oblast, ve které může být metodika rozšiřována v různých aspektech. Zaměřil jsem se zde na konvence při vývoji v rámci frameworku ASP.NET MVC, jako je struktura projektu či pojmenování jeho obsahu, díky kterým je mimo jiné vývoj velice usnadněn. Dále jsem pro několik vybraných produktů popsal tzv. osvědčené postupy při vývoji, které by se měly při vývoji dodržovat.

V průběhu tvorby úprav jsem se držel myšlenky metodiky MMSP, kterou je vytvoření rámce, nikoli podrobného návodu. Zaměřil jsem se na obecné principy, jež je nutné využít v každém projektu ASP.NET MVC a snažil jsem se vyhnout konkrétním implementacím, které by se mohly lišit v závislosti na projektu - každý vývojář má několik možností jejich řešení.

V poslední, šesté, kapitole této práce se věnuji ověření upravené metodiky MMSP na reálném příkladu, kterým je aplikace pro správu databáze antropologického oddělení Národního muzea. Jelikož jsem na vývoji pracoval sám, nebyl využit celý potenciál metodiky ve smyslu týmové koordinace a spolupráce. Snažil jsem se však co nejvíce držet definovaného procesu vývoje, kdy jsem iterativně vykonával jednotlivé úlohy s důrazem na dodržení pravidel stanovených v metodice.

Hlavním přínosem práce shledávám právě rozšířenou metodiku MMSP - ASP.NET MVC, která je publikována také v podobě webových stránek, aktuálně dostupných na adrese http://kitscm.vse.cz. Metodika by dle mého názoru mohla být využívána v rámci předmětu 4IT449 - Vývoj aplikací na platformě .NET, vyučovaného na VŠE v Praze, především při zpracovávání závěrečné semestrální práce. Studentům začínajícím s touto technologií by byla dobrou oporou vedoucí k správnému pochopení a implementaci webové aplikace. Metodika byla samozřejmě navržena nejen pro využití ve školních projektech, ale také v reálných projektech zabývajících se vývojem v technologii ASP.NET MVC. Pevně věřím, že by k využívání metodiky v reálných projektech přispělo právě její zařazení do výuky a následné zkušenosti ze strany studentů.

Pro další úpravu metodiky by bylo vhodné získání zpětné vazby z projektů, v rámci kterých byla využita. Bylo by například užitečné využít některých prvků z jiných úprav metodiky MMSP vzniklých v rámci diplomových prací na VŠE v Praze. Vzhledem k povaze metodiky, která má být jednoduchá a agilní, by však přidávání velkého množství obsahu nemuselo být přínosné. Místo toho by bylo vhodnější obsahem této diplomové práce doplnit některou z robustnějších metodik a případné rozšiřování provádět až poté.

Terminologický slovník

Anglický pojem	Zkratka	Význam (zdroj)
Active Server Pages	ASP	Skriptovací platforma společnosti Microsoft, určená pro dynamické zpracování webových stránek. Jedná se o předchůdce ASP.NET (Podešva, 2011).
Application Programming Interface	API	Rozhraní sloužící ke komunikaci mezi programy (MSDN, 2016).
Asynchronous JavaScript and XML	AJAX	Technologie umožňující zpracování dotazu na server a zobrazit výsledek, aniž by se načítala celá stránka (Freeman, 2013).
Cascading Style Sheets	CSS	Pravidla, která určují vzhled a formátování HTML dokumentu (Mittner, 2011)
Data redundancy		*„Redundantní data jsou hodnota, která je opakovaná v poli jako výsledek spoluúčasti pole při spojování dvou tabulek, nebo jako výsledek anomálie některého pole či tabulky“* (Hernandez, 2006).
Dynamic HTML	DHTML	Kombinace technologií používaných k tvorbě dynamických a interaktivních webových stránek (HTML, CSS, JS)
Eclipse Process Framework Composer	EPFC	Nástroj pro správu a údržbu metodiky (Šimko, 2015)
Framework		Kolekce abstraktních a konkrétních tříd a vztahů mezi nimi, která představuje základ systému (Buchalcevová, 2009).
Hypertext Markup Language	HTML	Značkovací jazyk tvořící základ všech webových stránek (ITnetwork.cz, 2015).
Hypertext Transfer Protocol	HTTP	Internetový protokol určený pro výměnu hypertextových dokumentů ve formátu HTML.
Hypertext Preprocessor	PHP	Skriptovací programovací jazyk určený určen pro tvorbu webových aplikací, či dynamicky generovaných stránek (Kosek, 1999).
JavaScript	JS	Skriptovací jazyk, který se využívá zejména k programování prezentační vrstvy webové aplikace (Mittner, 2011)

Anglický pojem	Zkratka	Význam (zdroj)
Model view controller	MVC	Softwarová architektura oddělující datový model, uživatelské rozhraní a řídící logiku do nezávislých objektů (Pecinovský, 2007).
Rational Unified Process	RUP	Komplexní metodika pro řízení projektů a tvorbu softwaru s využitím objektového přístupu založeného na UML (Buchalcevová a Stanovská, 2013).
Representational State Transfer	REST	Architektura webového rozhraní, která nám umožňuje přistupovat k datům a provádět nad nimi CRUD operace (ITnetwork.cz, 2015).
Relational database management system	RDBMS	(Relational database management system) – Systém pro správu relační databáze (Oppel, 2010).
Session		Prostředky, které umožňují řešit bezstavovost protokolu http (Mittner, 2011)
Strongly typed language		Silně typovaný jazyk provádí kontrolu typů a spolehlivě zjišťuje jejich chybné použití testů (Galloway et al., 2014).
Structured Query Language	SQL	Strukturovaný jazyk používaný pro práci s relační databází (Hernandez, 2006).
Tag		Části značek (elementů) v kódu značkovacího jazyka, např. HTML (Autor).
Unified Methody Architecture	UMA	Metamodel, který byl vytvořen za účelem sjednocení různých metod a procesních jazyků. (Šimko, 2015)
Unified Modeling Language	UML	Grafický jazyk pro vizualizaci, dokumentaci a konstrukci, definuje celou řadu modelů, kterými lze modelovat software z různých pohledů (Buchalcevová a Stanovská, 2013).
Uniform Resource Locator	URL	Způsob, jak jednoznačně zapsat umístění souboru na Internetu (ITnetwork.cz, 2015).
User interface	UI	Část aplikace sloužící pro interakci s uživatelem (Buchalcevová, 2009).
Web cache		Technologie pro dočasné uložení prvků webových stránek k snížení přenosu dat v síti (Huston, 1999).
Wireframe	WF	Představuje základní zobrazení všech komponent webové stránky a jejich vzájemné umístění. (Bank, 2014)

Použité informační zdroje

ASP.NET MVC [online]. Microsoft, 2016 [cit. 2016-06-11]. Dostupné z: http://www.asp.net/mvc

BANK, Chris a Waleed ZUBERI. The Guide to Wireframing [online], 2014. Dostupné z: https://www.uxpin.com/wireframing-hands-on-guide

BUCHALCEVOVÁ, Alena. Metodiky budování informačních systémů. Praha: Oeconomica, 2009. ISBN 978-80-245-1540-3.

BUCHALCEVOVÁ, Alena a Iva STANOVSKÁ. Příklady modelů analýzy a návrhu aplikace v UML. Praha: Oeconomica, 2013. ISBN 978-80-245-1922-7.

CONOLLY, Thomas, Carolyn E. BEGG a Richard HOLOWCZAK. Mistrovství - databáze: profesionální průvodce tvorbou efektivních databází. Brno: Computer Press, 2009. ISBN 978-80-251-2328-7.

DOBISÍKOVÁ, Miluše a Miloš ANDĚRA. Národní muzeum, Praha: průvodce: Přírodovědecké muzeum. Praha: Národní muzeum, 1999. ISBN 80-7036-052-6.

ECMA-334 Standard: C# Language Specification [online]. 2006. Ecma International [cit. 2016-10-02]. Dostupné z: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf

EPFP, Eclipse Process Framework Project [online]. The Eclipse Foundation, c2016. [cit. 2016-11-21]. Dostupné z: http://www.eclipse.org/epf/

FOWLER, Martin a Jim HIGHSMITH. 2001. The agile manifesto. Software Development, 9.8: 28-35.

FREEMAN, Adam. Pro ASP.NET MVC 5, 2013. Pro ASP.NET MVC 5. ISBN 1430265299.

GALLOWAY, Jon, Brad WILSON, David MATSON a K. Scott ALLEN. Professional ASP. NET MVC 5. Indianapolis, Indiana: John Wiley & Sons, Inc., 2014. ISBN 9781118794760.

GARRETT, Jesse James. The elements of user experience: user-centered design for the Web and beyond. 2nd ed. Berkeley, CA: New Riders, c2011. Voices that matter. ISBN 0321683684.

Google [online]. 2016 [cit. 2016-06-12]. Dostupné z: https://www.google.com

Grafická a multimediální laboratoř (GML): Web design a uživatelská rozhraní [online]. Praha: VŠE, 2014 [cit. 2016-07-31]. Dostupné z: http://gml.vse.cz/grantyaaktivity/oppa/vystupy-jednotlivych-kapitol-grantu/web-design-a-uzivatelska-rozhrani/

HARWOOD, Mike. Security Strategies in Web Applications and Social Networking. Sudbury, MA: Jones & Bartlett Learning, 2015. Jones & Bartlett learning information systems security & assurance series. ISBN 1284104354.

HERNANDEZ, Michael J. Návrh databází. Praha: Grada, 2006. Profesionál. ISBN 80-247-0900-7.

HUSTON, Geoff. Web Caching. Cisco, The Internet Protocol Journal [online]. 1999 [cit. 2016-11-23]. Dostupné z: http://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-2/ipj-archive/article09186a00800c8903.html

ITnetwork.cz [online]. Praha: David Čápka, 2015 [cit. 2016-06-11]. Dostupné z: http://www.itnetwork.cz/csharp/asp-net/mvc

KÁBRT, Jakub. Využití metodiky MMSP při vývoji IS v prostředí FileMaker. Praha, 2014. Diplomová práce. Vysoká škola ekonomická v Praze. Vedoucí práce doc. Ing. Alena Buchalcevová, Ph.D.

KOSEK, Jiří. PHP: tvorba interaktivních internetových aplikací: podrobný průvodce. Praha: Grada, 1999. ISBN 80-7169-373-1.

KVAPIL, Jiří. Webová aplikace založená na frameworku ASP.NET MVC 3/RAZOR. Brno, 2012. Bakalářská práce. Vysoké učení technické v Brně. Vedoucí práce Ing. Rudolf Kajan,

MEYER, Bertrand. Agile!: the good, the hype and the ugly. Aufl. 2014. Cham: Springer International Publishing, 2014. ISBN 9783319051550.

MITTNER, Jan. Metodika pro vývoj webových aplikací. Praha, 2011. Diplomová práce. Vysoká škola ekonomická v Praze. Vedoucí práce doc. Ing. Alena Buchalcevová, Ph.D.

MMSP [online]. Praha. Rejnková Petra, 2011 [cit. 2016-06-14]. Dostupné z: http://mmsp.czweb.org/

MSDN (Microsoft Developer Network) [online]. Microsoft, 2016 [cit. 2016-08-21]. Dostupné z: https://msdn.microsoft.com

NOVOTNÝ, Roman. Rozšíření metodiky MMSP v oblasti analýzy a návrhu testování. Praha, 2015. Diplomová práce. Vysoká škola ekonomická v Praze. Vedoucí práce doc. Ing. Alena Buchalcevová, Ph.D.

NÝVLT, David. Prototypování při vývoji softwaru. Praha, 2012. Diplomová práce. Vysoká škola ekonomická v Praze. Vedoucí práce doc. Ing. Alena Buchalcevová, Ph.D.

OpenUP 1.5.1.4 [online]. The Eclipse Foundation, 2012 [cit. 2016-06-12]. Dostupné z: http://epf.eclipse.org/wikis/openup/

OPPEL, Andrew J. Data modeling: a beginner's guide. New York: McGraw-Hill, c2010. ISBN 0071623981.

PALOVSKÁ, Helena. Datové modelování. Krokodýlovy databáze [online]. Praha, 2011 [cit. 2016-07-14]. Dostupné z: http://krokodata.vse.cz/DM

PECINOVSKÝ, Rudolf. Návrhové vzory: [33 vzorových postupů pro objektové programování]. Brno: Computer Press, 2007. ISBN 978-80-251-1582-4.

PODEŠVA, Tomáš. Vývoj webových aplikací v Microsoft ASP.NET MVC a WebForms. Olomouc, 2011. Bakalářská práce. Univerzita Palackého v Olomouci. Vedoucí práce Ing. Jiří Hronkov.

REJNKOVÁ, Petra. Lokalizace a přizpůsobení metodiky OpenUP. Praha, 2011. Diplomová práce. Vysoká škola ekonomická v Praze. Vedoucí práce Doc. Ing. Alena Buchalcevová, Ph.D.

RUP for Small Projects [online]. 2006 [cit. 2016-06-12]. Dostupné z: https://kitscm.vse.cz/RUP/SmallProjects/index.htm

ŠIMKO, David. Lokalizace Vstupního profilu normy ISO/IEC 29110 do češtiny a jeho publikace v Eclipse Process Framework Composer. Praha, 2015. Diplomová práce. Vysoká škola ekonomická v Praze. Vedoucí práce doc. Ing. Alena Buchalcevová, Ph.D.

TechFundaTM.com [online]. India, 2016 [cit. 2016-06-12]. Dostupné z: http://techfunda.com/howto/asp-net-mvc

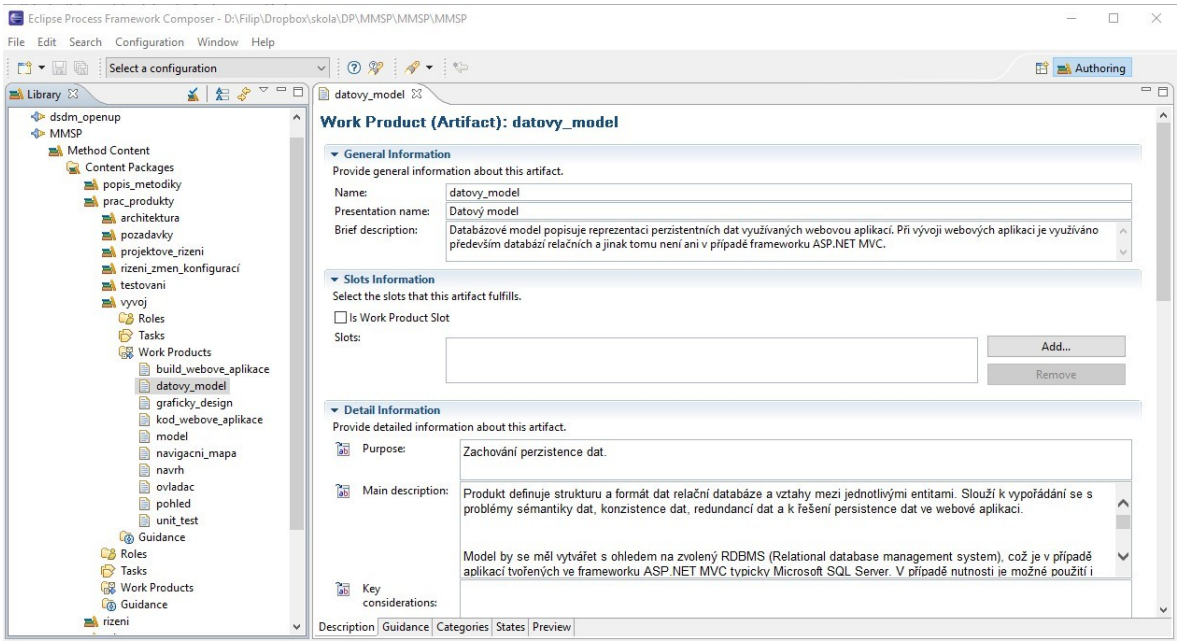
TechNet: Resources and Tools for IT Professionals [online]. Microsoft, 2016 [cit. 2016-10-23]. Dostupné z: https://technet.microsoft.com

VELEMÍNSKÝ, Filip. Návrh databáze pro antropologické oddělení Národního muzea. Praha, 2014. Bakalářská práce. Vysoká škola ekonomická v Praze. Vedoucí práce RNDr. Helena Palovská, Ph.D.

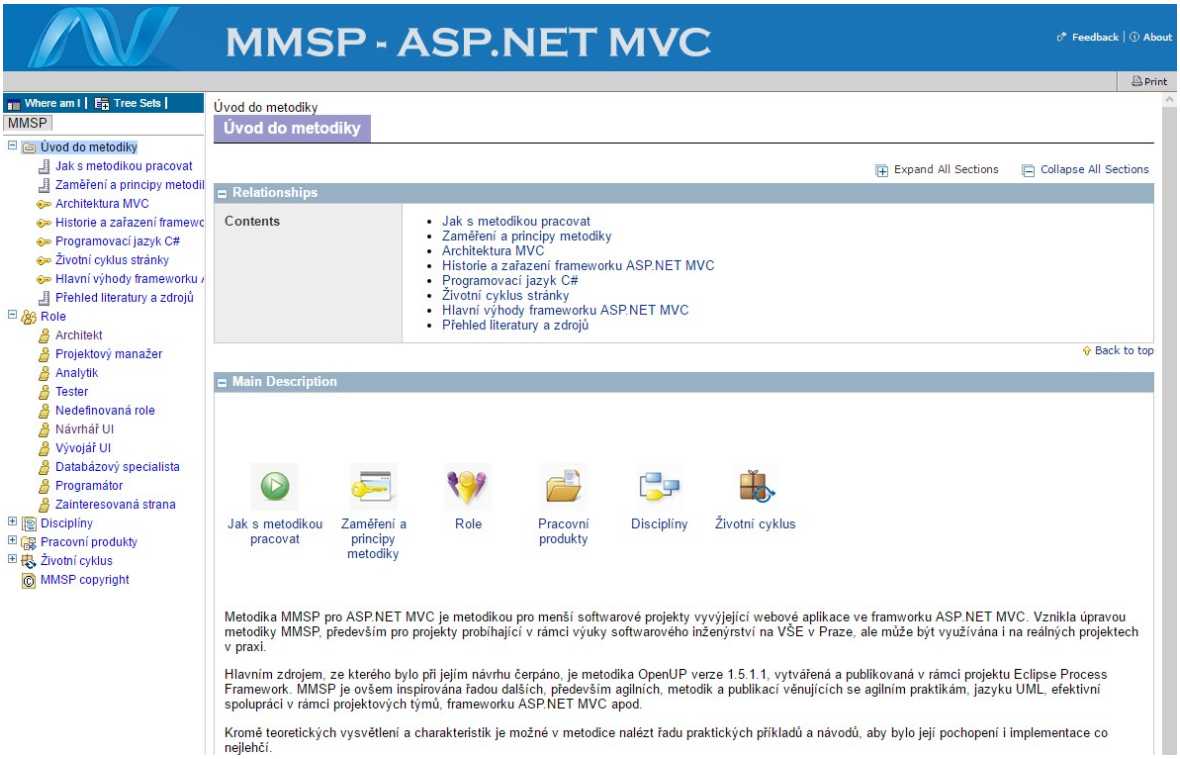
VŠE. Elektronické informační zdroje dostupné na VŠE [online]. 2016 [cit. 2016-06-12]. Dostupné z: http://www.vse.cz/zdroje/

WAKE, Bill. 3A – Arrange, Act, Assert. Exploring Extreme Programming [online]. 2011 [cit. 2016-09-04]. Dostupné z: http://xp123.com/articles/3a-arrange-act-assert/

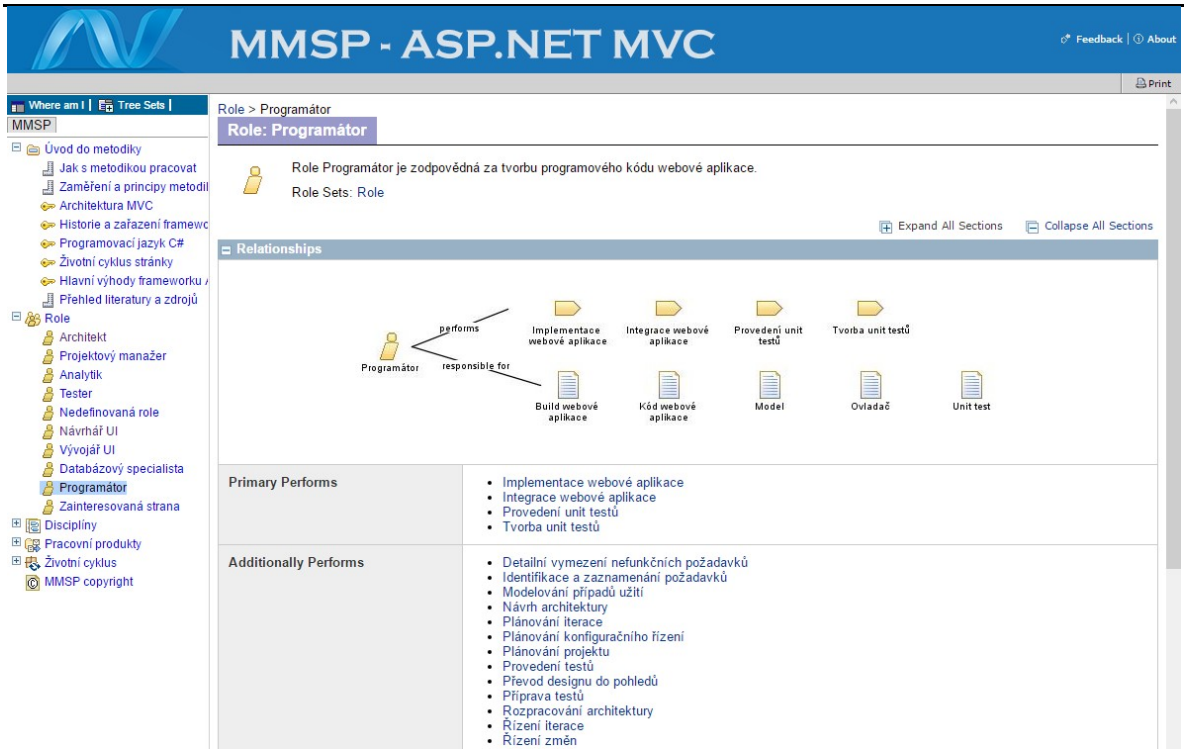
Příloha A: Snímky publikované metodiky MMSP – ASP.NET MVC



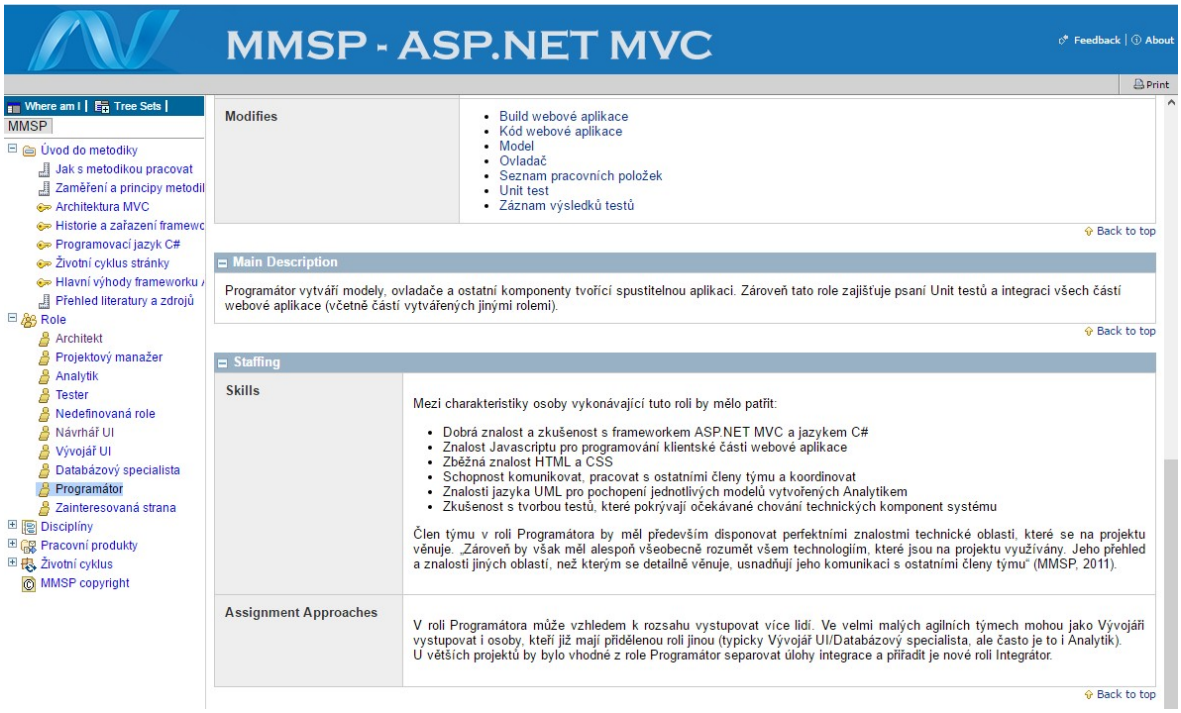
Obrázek A1: Úprava produktu Datový model v EPFC



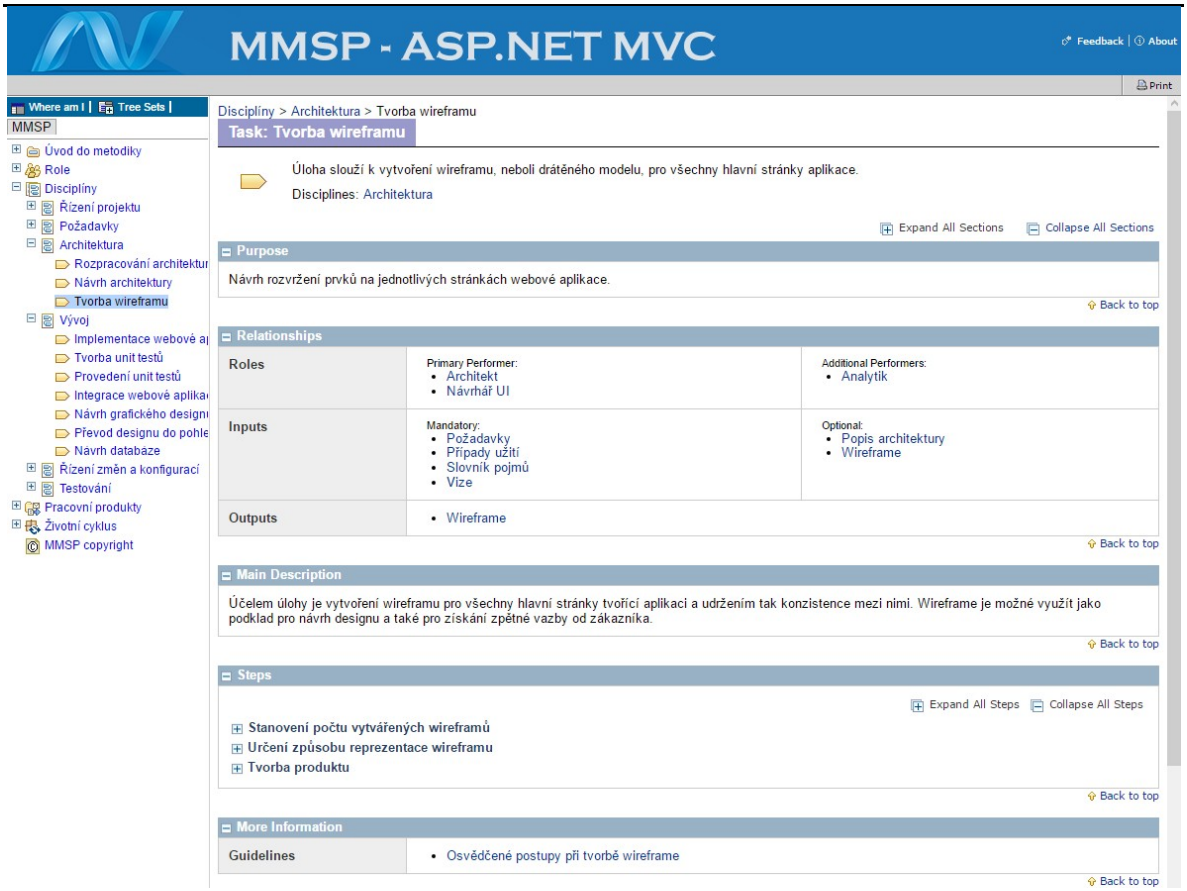
Obrázek A2: Úvodní obrazovka metodiky



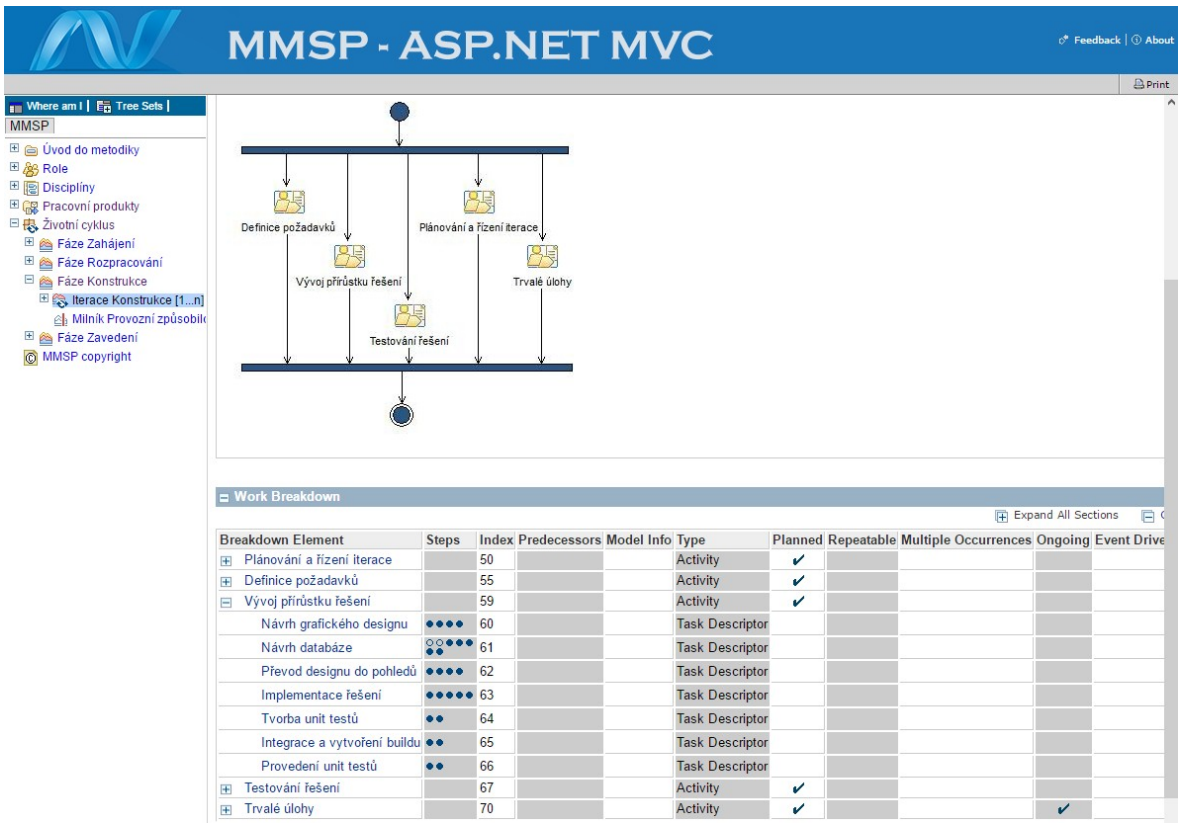
Obrázek A3: Role Programátor – zobrazení vztahů a vykonávaných úloh



Obrázek A4: Role Programátor – zobrazení Pracováních produktů, které modifikuje; požadovaných dovedností a možností přiřazení role.



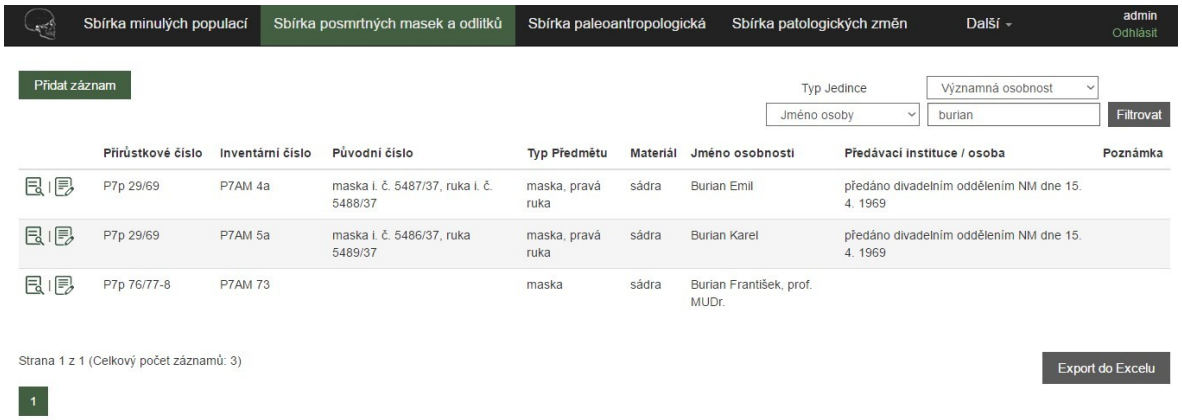
Obrázek A5: Úloha Tvorba wireframu



Obrázek A6: Životní cyklus metodiky – fáze konstrukce

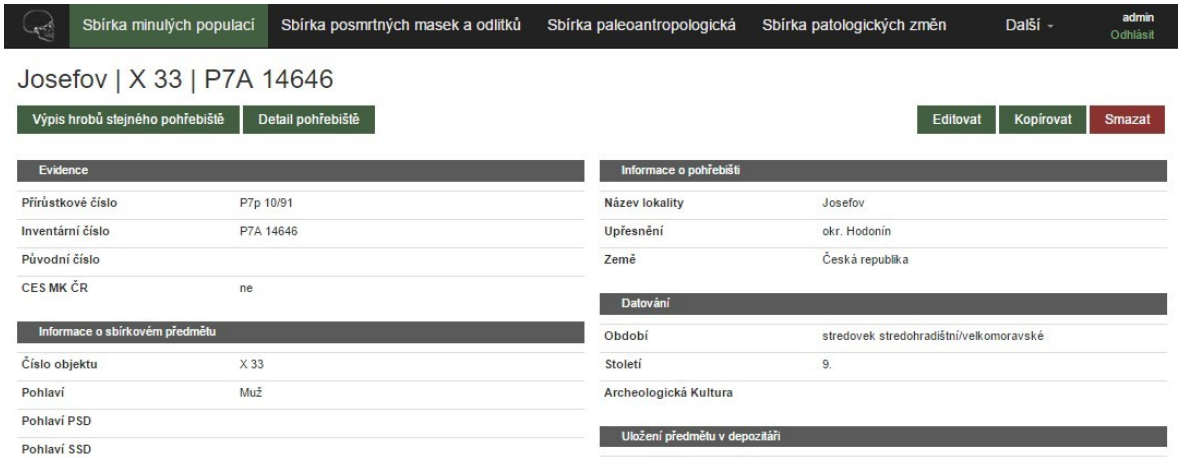
Příloha B: Snímky vytvořené aplikace

Obrázek B1: Hlavní obrazovka aplikace

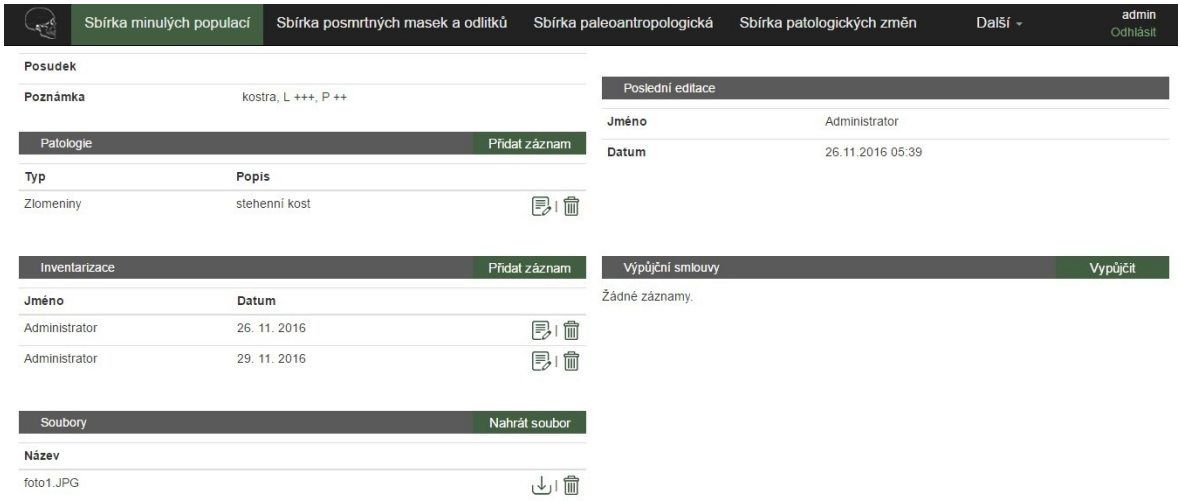


Obrázek B2: Seznam záznamů sbírky posmrtných masek a odlitků, filtrované podle jména významné osobnosti

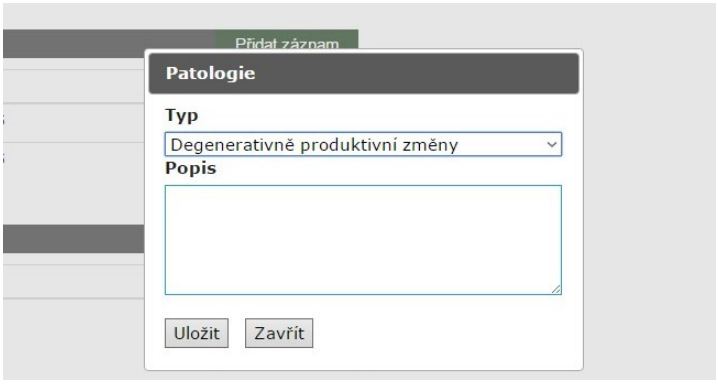
Obrázek B3: Seznam záznamů sbírky paleoantropologické, zachycení fixní hlavičky tabulky při posunu níže



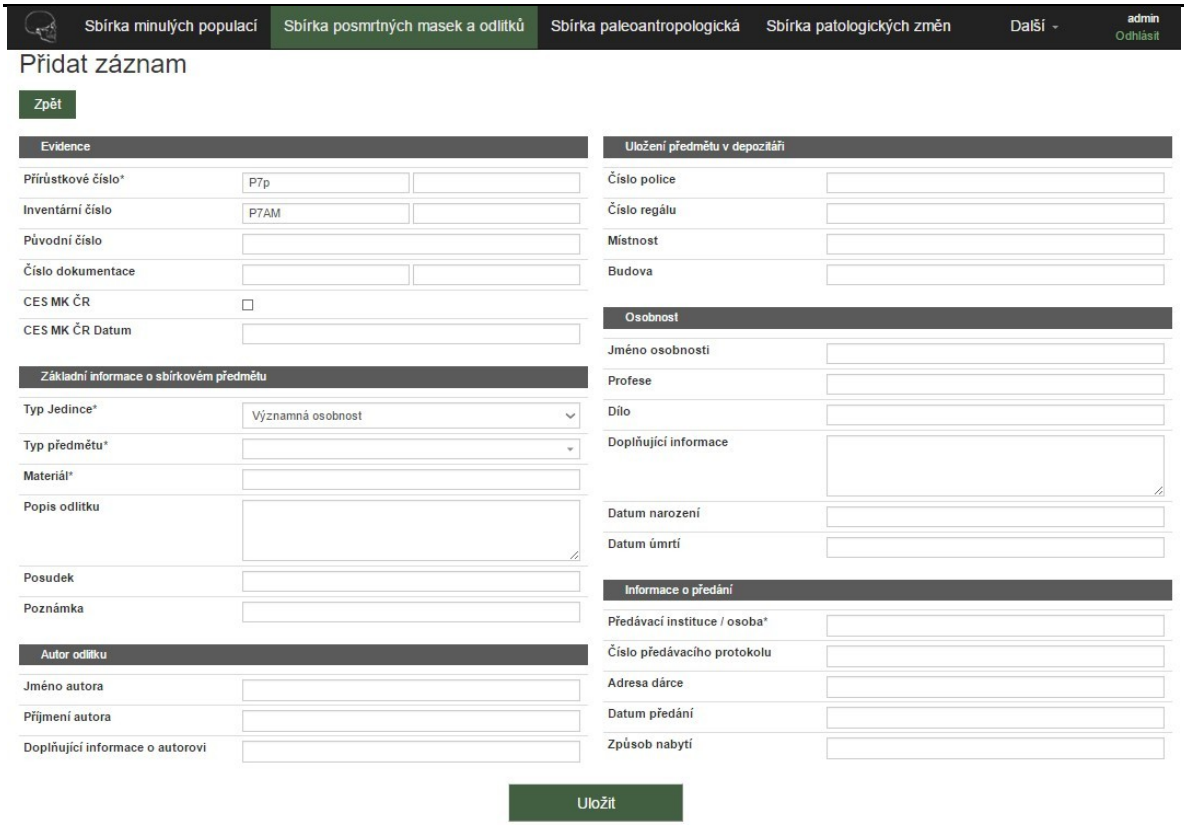
Obrázek B4: Detail záznamu hrobu ze sbírky minulých populací – základní atributy



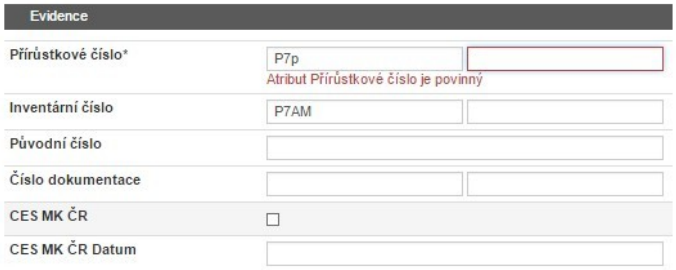
Obrázek B5: Detail záznamu – atributy ve vazbě 1:N



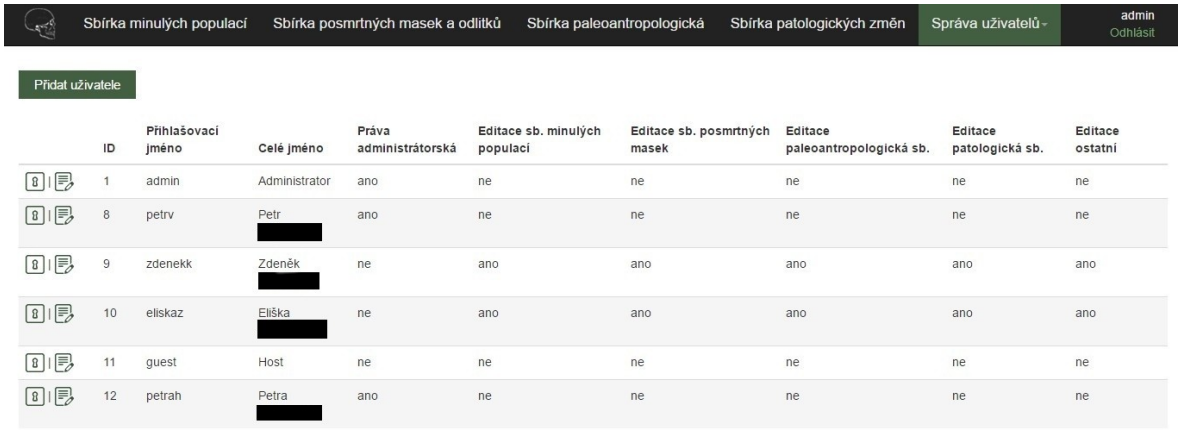
Obrázek B6: Přidání nové patologie k záznamu sbírky



Obrázek B7: Formulář pro přidání nového záznamu do sbírky posmrtných masek a dolitků



Obrázek B8: Validace povinného pole



Obrázek B9: Správa uživatelů a jejich oprávnění v databázi