Univerzita Hradec Králové

Fakulta informatiky a managementu

Implementace sportovního portálu v .NET

Diplomová práce

Autor: Bc. Eduard Hloušek

Studijní obor: Aplikovaná informatika – magisterský navazující

Vedoucí práce: Ing. Pavel Kříž, Ph.D.

Hradec Králové duben 2015

Prohlášení

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne Bc. Eduard Hloušek

Poděkování

Rád bych poděkoval vedoucímu této práce Ing. Pavlu Křížovi, Ph.D. za odborné vedení a užitečné rady.

Anotace

Tato práce si klade za cíl prozkoumat a popsat technologii ASP.NET, která se používá pro vývoj webových aplikací. Práce je rozdělená do dvou hlavních částí. První část rozebírá historii webového vývoje a popisuje vývoj aplikačního rámce .NET, kterého je ASP.NET součástí. Druhá část práce se věnuje vývoji reálného projektu se sportovní tematikou. Popisuje návrh aplikace a následnou implementaci jejích hlavních aspektů, přičemž důraz je kladen na použití moderních technologií a postupů.

Klíčová slova: ASP.NET, MVC, sportovní portál, webový vývoj

Annotation

This diploma thesis “Implementation of sports portal in .NET” is focused on exploring and describing ASP.NET technology used for development of web applications. The thesis is divided into two main parts. The first part examines the history of web development and describes the progress of .NET framework which includes ASP.NET. The second part deals with the development of a real project with sports topic. It describes the design and the following implementation of its main aspects. It emphasizes the usage of modern technologies and techniques.

Keywords: ASP.NET, MVC, sports portal, web development

Obsah

1 Úvod 1

2 Představení ASP.NET a jeho moderní vývoj 3

2.1 Co je ASP.NET? 3

2.2 Webová tvorba před ASP.NET 4

2.3 Příchod a vývoj ASP.NET 6

2.3.1 ASP.NET 1.0 a 1.1 6

2.3.2 ASP.NET 2.0 7

2.3.3 ASP.NET 3.5 7

2.3.4 ASP.NET 4.0 11

2.3.5 ASP.NET 4.5 12

2.3.6 ASP.NET 4.5.1 12

2.4 Vývojové modely 12

2.4.1 ASP.NET Web Pages 13

2.4.2 ASP.NET Web Forms 13

2.4.3 ASP.NET MVC 14

3 Požadavky na aplikaci 19

3.1 České portály 19

3.1.1 FotbalPortal.cz 19

3.1.2 eFotbal.cz 21

3.1.3 EuroFotbal.cz 23

3.2 Funkční požadavky 25

3.3 Technologické požadavky 27

4 Návrh aplikace 29

4.1 Diagram případů užití 29

4.2 Entity-relationship model 33

5 Implementace 35

5.1 Visual Studio 2012 35

5.2 Visual Studio 2013 37

5.3 Visual Studio Online 37

5.4 Nový projekt 37

5.5 Databáze 42

5.5.1 MS SQL Server 2012 43

5.5.2 Entity Framework ... 44
5.6 Model, View, Controller ... 46
5.7 Migrations ... 49
5.8 Autentizace a autorizace ... 51
5.9 Menu ... 56
5.10 Články a komentáře ... 57
5.11 Evidence klubů a hráčů ... 62
5.12 Tabulky a hráčské statistiky ... 65
5.13 Videopříspěvky ... 76
5.14 Názory čtenářů ... 78
5.15 Výsledky živě ... 79
5.16 Nasazení ... 80
5.17 GIT ... 81
6 Zhodnocení 83
7 Závěr 85
8 Použitá literatura 86
Příloha 1 – Obsah CD 92

1 Úvod

Webový vývoj urazil za posledních 20 let obrovský kus cesty. Nepochybně za to mohl výrazný rozmach internetu, který neustále nutí inženýry vyvíjet nové technologie, aby byli vývojáři a uživatelé spokojeni. V rámci historie lidstva je to několik okamžiků zpět, kdy byly webové stránky jednoduché, statické a tvořeny výhradně pomocí značek HTML jazyka. HTML jazyk v modernější podobě přežil až do dnešních dnů, protože pro definici rozložení webové stránky zatím nic lepšího neznáme, ale používá se naprosto jiným způsobem. Dříve programátoři psali webové stránky výhradně pomocí HTML značek. Dnes se tomu tak děje pouze částečně, anebo vůbec. Postupem času byly žádány zajímavější a na pohled hezčí webové stránky. Příchodem kaskádových stylů (CSS) bylo umožněno oddělit strukturu webové stránky od jejího vzhledu a vývojem technologií jako PHP a JavaScript se webovým stránkám přidalo na dynamičnosti. Postupem času bylo možné tvořit webové stránky stále více a více programovacími jazyky. Dnes má vývojář tolik možností jak vytvořit webové stránky, že snad ani není možné vyzkoušet je úplně všechny. Většinou si každý najde menší počet oblíbených technologií, ve kterých programuje a prohlubuje své znalosti.

Jak již bylo zmíněno, internet zažil obrovský rozmach. A proto se o tvorbu webových stránek začali zajímat i vývojáři, kteří do té doby programovali pouze desktopové aplikace. Aby se nemuseli učit úplně nové postupy, výrobci softwaru začali vydávat aplikační rámce (frameworky), které se snaží usnadnit práci nejen jim, ale i stávajícím webovým vývojářům a všem jejich následovníkům. Šlo jim o to, aby co nejvíc spojili tvorbu webových a desktopových aplikací. Aplikačních frameworků pro vývoj webových aplikací vzniklo velké množství. Namátkou můžeme zmínit *Apache Struts*, *Django*, *Ruby on Rails*, *Spring MVC* nebo *Zend*. V této chvíli se dostáváme do bodu, kdy jedním z oněch frameworků je i ASP.NET, který je součástí většího aplikačního rámce .NET. Autorem je softwarový gigant Microsoft.

Tato diplomová práce se bude zabývat tvorbou webové aplikace pomocí silné technologie ASP.NET. V úvodu této práce se čtenář seznámí s tím, co vlastně technologie ASP.NET je a jaký byl její historický vývoj. Jako hlavní cíl si tato práce klade zaměřit se na aktuální trendy v tvorbě webových aplikací na platformě .NET.

V dalších kapitolách se zaměří na praktickou tvorbu webové aplikace, kterou se bude snažit projít od nastolení požadavků, přes návrh až k samotné implementaci. Objektem vývoje v rámci této práce bude sportovní portál, protože autor sám patří mezi sportovní příznivce a jeho záměrem je vytvořit aplikaci, která bude vhodná pro ostré nasazení. Závěrečné kapitoly zhodnotí vytvořenou práci a pokusí se nastínit jakým směrem by se mohl vývoj ubírat jak obecně, tak na platformě .NET.

2 Představení ASP.NET a jeho moderní vývoj

Tato teoretická kapitola se bude zabývat problematikou programování webových aplikací technologií ASP.NET. Čtenář se dozví, co si má pod touto zkratkou představit a jak se liší od pojmu .NET Framework. Kapitola se také bude chronologicky zabývat historickým vývojem této technologie od jejích počátků až do současné moderní podoby.

2.1 Co je ASP.NET?

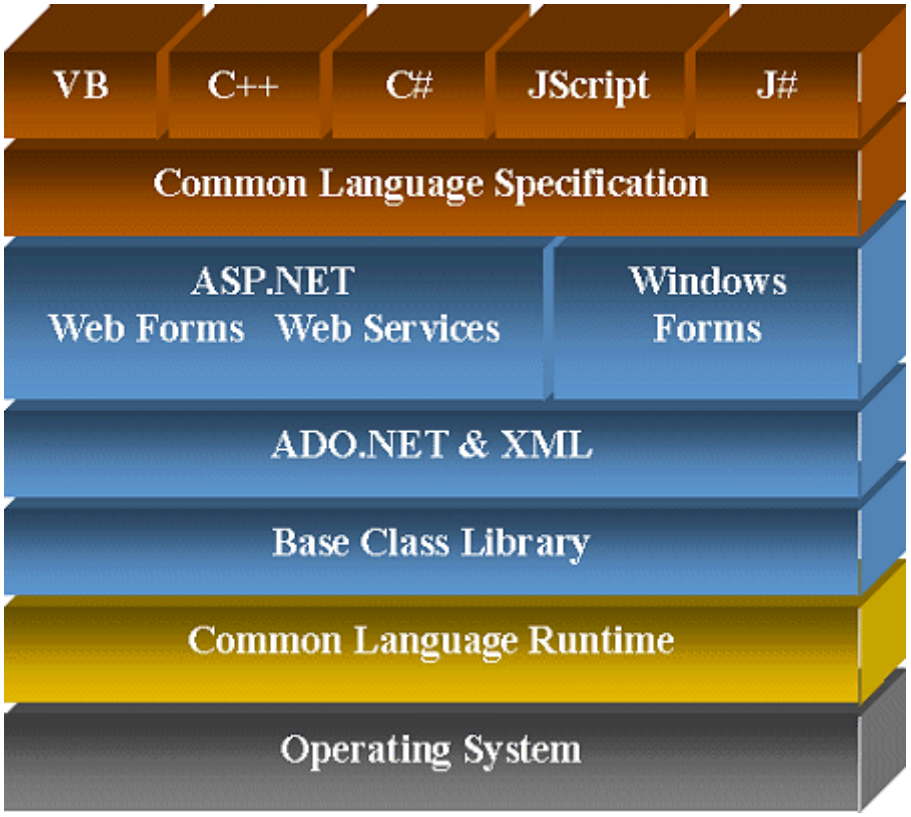
Předně bude vymezeno co přesně se skrývá pod zkratkou ASP.NET a co je pod tím si třeba představit. Ačkoliv je název odvozen od starších ASP (Active Server Pages), jsou obě technologie odlišné. ASP je „pouze“ skriptovací platforma pro generování dynamických webových stránek na straně serveru vyvinutá společností Microsoft. Častou chybou méně znalého člověka nebo začínajícího vývojáře, je domněnka, že ASP je programovací jazyk. Není tomu tak. Pro kódování ASP stránek se používají skriptovací jazyky. Nejpoužívanějšími jsou VBScript (Visual Basic Script Edition) a JScript vyvinuté taktéž Microsoftem.

Oproti tomu pod pojmem ASP.NET se skrývá silná technologie pro vývoj webových aplikací, která je součástí systému Microsoft .NET Framework. Tedy ani v případě technologie ASP.NET se nejedná o žádný programovací jazyk. Výkonný kód lze psát v několika jazycích (např.: C# a Visual Basic .NET). Nadřazený .NET Framework, pod který ASP.NET patří, je označení pro celý soubor technologií, které se používají nejen pro webový vývoj. ASP.NET je tedy jeho komponenta, která je jednou z nejpoužívanějších.

Další komponenty systému Microsoft .NET Framework

- Windows Communications Foundation (WCF) – aplikační rozhraní, neboli API (Application Programming Interface), pro tvorbu webových služeb. Je to základní infrastruktura pro tvorbu servisně orientovaných aplikací.
- Windows Workflow Foundation (WF) – technologie, která umožňuje modelování procesů na platformě .NET.

- Windows Presentation Foundation (WPF) – technologie, která umožňuje oddělit funkčnost a vzhled aplikace. Slouží k tvorbě poutavého uživatelského rozhraní.
- Language Integrated Query (LINQ) – integrovaný jazyk, který umožňuje a usnadňuje dotazování nad jakýmikoliv daty.



Obrázek 2.1: Architektura .NET Frameworku [36]

2.2 Webová tvorba před ASP.NET

Historii webu začal psát Timothy Bernes-Lee, toho času ředitel konsorcia W3C (*World Wide Web Consortium*), který vynalezl *World Wide Web* – běžně označován jako „web“. Pod tímto heslem se skrývá označení pro aplikace internetového protokolu HTTP. Hlavní činností konsorcia je vývoj protokolů a směrnic, které mají zajistit dlouhodobý rozvoj webu. Jejich standardy hrají klíčovou roli ve fungování celého webu. [2]

Po zřízení protokolu HTTP nebylo pro vývojáře jednoduché zajistit, aby spolu webové aplikace komunikovaly a navzájem se vyhledávaly. Pro podpoření jejich

úspěšné práce byly vyvinuty různé standardy. Například jazyky HTML a XML. Pomocí HTML je možné popsat, jak má počítač zobrazit kompilované dokumenty na jakékoliv platformě. Díky standardu XML je možné vytvořit sadu pravidel pro výměnu dat nezávisle na platformách. Ve stejný okamžik ale výrobci softwaru jako IBM, Sun Microsystems nebo Microsoft, čelili i svým vlastním výzvám. Potřebovali vyvinout nejen vlastní programovací jazyky, které by komunikovaly s webem, ale také vlastní pracovní rámce. Ty by vývojářům dovolily navrhovat architekturu a poté vyvíjet a rozšiřovat jejich aplikace. To vše co nejsnadnějším způsobem. Příchodem ASP.NET 1.0 započala nová kapitola ve vývoji webových aplikací. Microsoft technologií .NET nabídl integrovanou skupinu komponent, která obsahuje budování webu (HTTP a značkovací jazyky) a vyzkoušenou metodologii, která se orientuje na objekty.

Starší technologie pro vývoj webových aplikací používaly skriptovací jazyky a proprietární konvence značkování. Většina technik pro webový vývoj spouštěla aplikace nebo komponenty na serveru pomocí tzv. *hooks* (funkce se vyvolá při spuštění určité události – „zahákne se“). Vývojáři postrádali moderní a integrovaný pracovní rámec pro vývoj webových aplikací, protože většina pracovních rámců spadala do dvou kategorií:

1. Skripty interpretované nějakým zdrojem na serveru.
2. Oddělené mini-aplikace, které se voláním vykonají na straně serveru.

Do první kategorie patří například klasické ASP (*Active Server Pages*) nebo *ColdFusion* (značkovací programovací jazyk). Při tomto přístupu vývojář vytvoří soubor se skriptem, který obsahuje vložený kód. Tento soubor zpracuje jiná komponenta, která střídavě zpracovává HTML a vložený kód. Nedostatkem takto skriptované aplikace oproti kompilované je ten, že se vykonává pomaleji. V tomto přístupu k webovému vývoji navíc chybí možnost řídit nastavení bezpečnosti a také obsahuje neefektivní využívání systémových zdrojů.

Druhý přístup, který využívá například Perl přes CGI (*Common Gateway Interface*), způsobuje jiné problémy. Při této činnosti pouští server oddělenou aplikaci, která zpracovává požadavek klienta. Aplikace vykoná kód, dynamicky vytvoří HTML a to je odesláno zpět ke klientovi. Oproti prvnímu způsobu jsou tyto aplikace vykonávány rychleji, ale jejich nedostatkem je, že spotřebují více paměti. Webový server totiž musí

oddělenou instanci aplikace vytvořit pro každý klientův požadavek. Tyto aplikace jsou nejen špatně integrovatelné s jinými komponentami, ale také se obtížně píšou a ladí.

ASP.NET je tedy více než prostým vývojem jednoho nebo druhého typu aplikace. Přichází totiž s úplně novým vývojovým modelem, který je odlišný v tom, že je integrovaný se svým pracovním rámcem. ASP.NET je součástí .NET Frameworku a je spravován jeho *runtimem* (knihovna potřebná k běhu programu). ASP.NET tedy maže dělicí čáru mezi vývojem webových a desktopových aplikací, protože do webového vývoje přináší nástroje a technologie, které dříve používali pouze vývojáři desktopových aplikací. [1]

2.3 Příchod a vývoj ASP.NET

2.3.1 ASP.NET 1.0 a 1.1

První verze technologie ASP.NET nesla označení 1.0 a byla vydána na počátku roku 2002. Svým odlišným a revolučním přístupem se stala tak populární, že si její licenci obstaraly tisíce komerčních webových serverů ještě ve fázi beta. Prakticky současně byl také vydán vývojový nástroj Visual Studio .NET.

ASP.NET sice vyznává jiný přístup, ale dobře známé principy týkající se tvorby webových stránek stále platí. Vývojář může zpracovávat HTML, používat JavaScript, tvořit komponenty pro zapouzdření programové logiky, nebo přizpůsobovat aplikace pomocí různých konfiguračních voleb. V pozadí však tato technologie pracuje jinak než tradiční skriptovací technologie jako ASP nebo PHP a je také ambiciózněji než JSP.

Několik základních rysů, ve kterých se ASP.NET liší od dřívějších vývojových platforem:

- Nabídka úplného, objektově orientovaného programovacího modelu, který obsahuje architekturu, řízenou událostmi a založenou na ovládacích prvcích. To podporuje zapouzdření a znovupoužití kódu.
- Umožňuje psát kód ve velkém množství programovacích jazyků. Některé jsou podporované a pro některé je potřeba kompilátor od jiných výrobců.

- Slouží jako platforma pro tvorbu webových služeb. Webové služby jsou opětovně využitelné jednotky kódu, které mohou volat jiné aplikace.
- Klade důraz na vysoký výkon. Stránky a komponenty se kompilují na požádání, a proto se neinterpretují pokaždé, když se použijí. ASP.NET má také vyladěný sytém přístupu k datům a jejich flexibilní ukládání do *cache*. Díky tomu je možné zvýšit výkon v případě potřeby. [1]

Rok po vydání první verze se objevila nová s označením 1.1, která vyšla společně s operačním systémem Windows Server 2003 a s vývojovým nástrojem Visual Studio .NET 2003. V této verzi přibyla nejen automatická validace vstupů, ale také nová technologie Mobile Controls, která nahrazuje starší technologii Microsoft Mobile Internet Toolkit a umožňuje vyvíjet webové aplikace pro mobilní zařízení. Tyto aplikace využívají mobilní telefon jako tenký klient a běží na serverech. [4]

2.3.2 ASP.NET 2.0

O dva roky později, v říjnu 2005, byla vydána verze 2.0, která přinesla spoustu novinek. Byla vydána nejen s novou verzí Visual Studia, ale také SQL Serveru. Samotná technologie přinesla novinky, které usnadňují práci vývojářům. Jedná se například o snazší správu uživatelů a uživatelských rolí. Také podpora personalizace pro uživatele, témat, skinů a *master pages*, které umožňují jednodušší rozložení webové stránky. Další novinkou je technologie Web Parts, která nabízí integrovanou sadu ovládacích prvků pro vytváření webů, umožňujících uživateli upravovat obsah, vzhled nebo chování stránek přímo z prohlížeče. Nová verze přinesla nové komponenty pro manipulaci s daty (*GridView*, *FormView*, *DetailsView*) a také pro deklaraci datových spojení (*SqlDataSource*, *ObjectDataSource*, *XMLDataSource controls*). Mezi další novinky patří navigační ovládací prvky, techniky pro lokalizaci nebo podpora 64-bitových procesorů. [5][6]

2.3.3 ASP.NET 3.5

Přesně o rok později přišel Microsoft s další aktualizací .NET Frameworku s označením 3.0, která přinesla komponenty a technologie WPF, WF, WCF, které byly zmíněny již dříve. V samotném ASP.NET nic nového nebylo a tak ASP.NET 3.0 neexistuje.

Již tradičně o rok později v listopadu 2008 se objevila nová verze 3.5, která přinesla novinky hlavně ze dvou oblastí – LINQ a Ajax. Kromě nových rysů se společně s ní vydaly i nové verze nástrojů Visual Studio a Windows Serveru. O technologii LINQ, která je dostupná nejen v ASP.NET, ale v celém .NET Frameworku, bylo zde již napsáno, že je to integrovaný jazyk, který umožňuje a usnadňuje dotazování nad jakýmikoliv daty. Protože existují různé druhy dat, existují také různé varianty technologie LINQ:

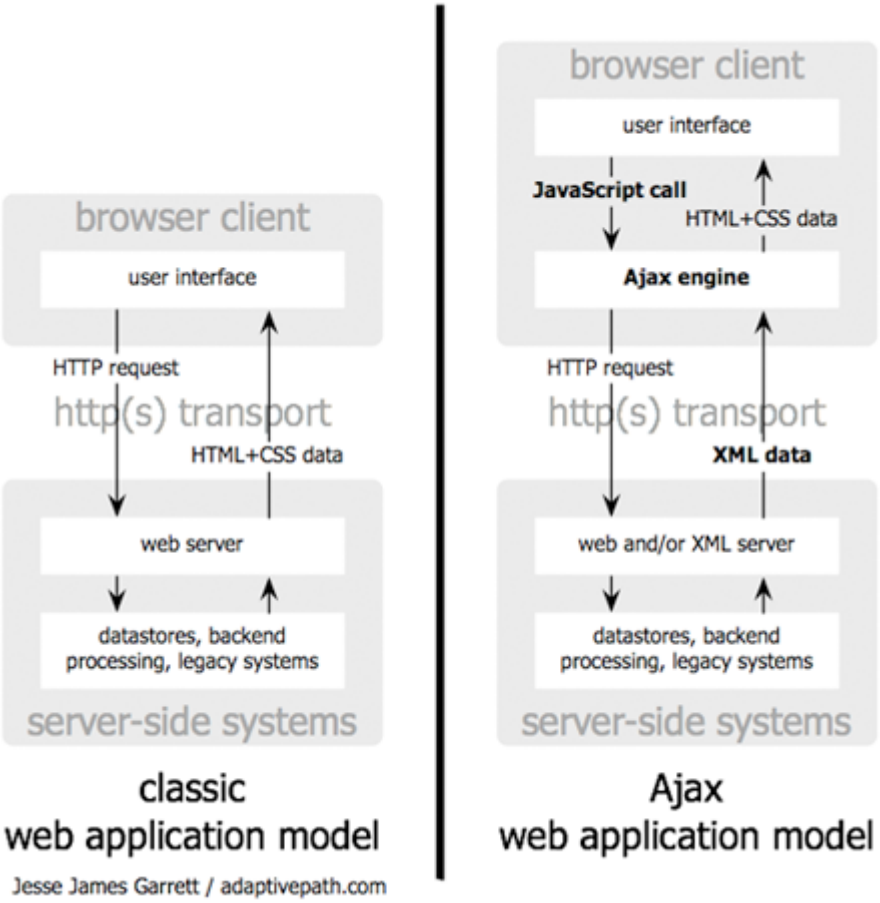
- LINQ to Objects – vykonává dotazy nad kolekcemi objektů
- LINQ to DataSet - vykonává dotazy na objekty sady dat (DataSet)
- LINQ to XML – pracuje s daty v XML souborech
- LINQ to SQL – používá syntaxi LINQ pro práci s databází SQL Serveru [7]

Již zmíněnou novinkou ve verzi 3.5 byla sada rozšíření pod názvem ASP.NET AJAX, která implementovala technologii Ajax (Asynchronous JavaScript and XML). První zmínka o technologii Ajax se objevila v roce 2005 v článku Jesse Jamese Garretta - Ajax: A New Approach to Web Applications [9].

Ajax ve skutečnosti není jedna technologie, ale soubor několika technologií, které dohromady tvoří silný nástroj a dělají webové stránky zajímavější a interaktivnější.

Ajax zahrnuje:

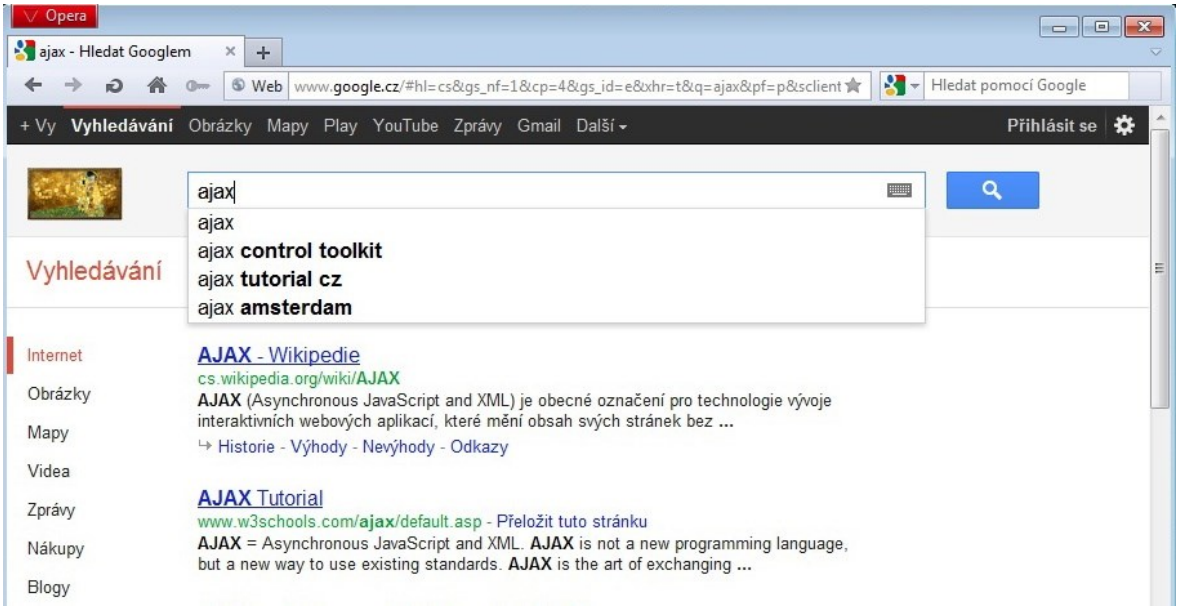
- Technologie HTML a CSS pro prezentaci údajů uživateli
- Dynamické interakce a zobrazení prostřednictvím Document Object Modelu (DOM)
- Výměnu dat prostřednictvím XML a XSLT
- Asynchronní komunikaci webového prohlížeče a serveru pomocí objektu XMLHttpRequest
- JavaScript, který to vše spojuje a zajišťuje chod celého soukolí zmíněných technologií



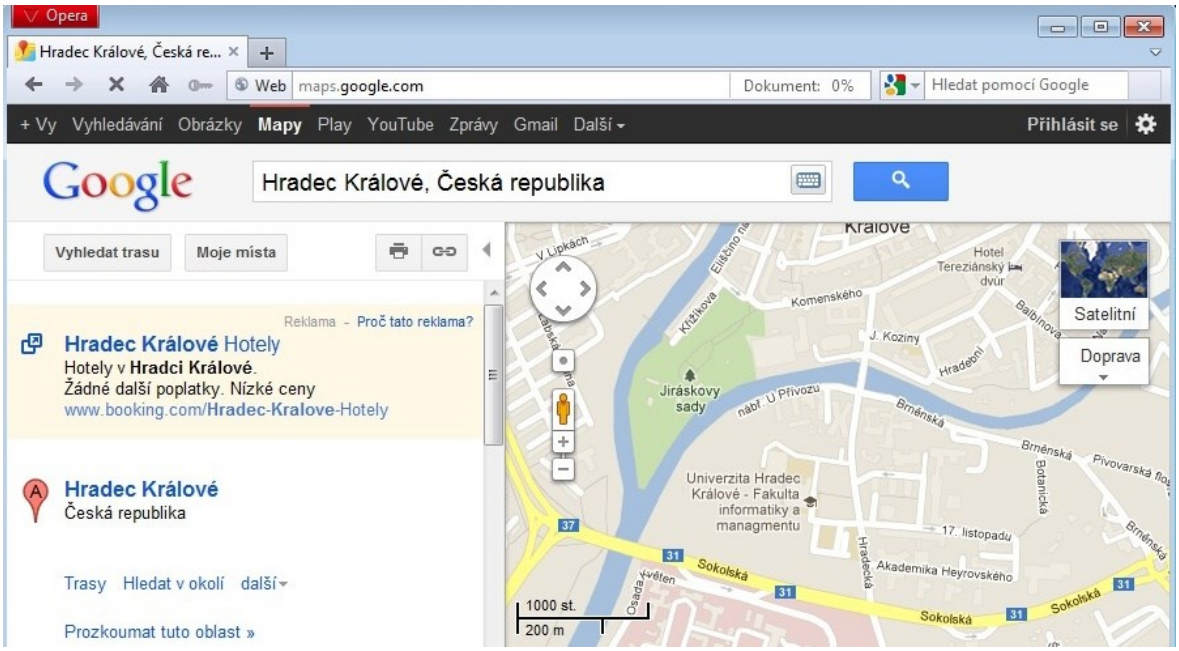
Obrázek 2.2: Zpracování požadavků ve webové aplikace bez Ajaxu (vlevo) a s Ajaxem (vpravo) [9]

V čem je vlastně Ajax odlišný? Webové stránky, které nepoužívají Ajax, se chovají způsobem „příkaz-načtení-příkaz-načtení“, kdy uživatel klikne na odkaz nebo tlačítko, server zpracuje jeho požadavek, zašle odpověď a webová stránka se načte znovu v aktualizované podobě. Ajax engine (na obrázku výše v pravé části) přidává vrstvu mezi uživatele a server. Ajax díky JavaScriptu neustále naslouchá uživatelským požadavkům a aktualizuje webovou stránku, která ve skutečnosti není kompletně znovu načtená, protože Ajax čeká na další příkazy. K úplnému pochopení je dobré uvést několik příkladů. S Ajaxem se nejspíše setkal už každý uživatel internetu. Stačí si v prohlížeči otevřít vyhledávač Google nebo český Seznam a do vyhledávacího pole zadat několik znaků hledaného slova. Webová stránka nabídne možná slovní spojení, aniž by se sama aktualizovala. Děje se to právě díky asynchronní činnosti Ajax enginu, který například Google využívá ve svém našeptávači a během psaní nabízí uživateli hledané výrazy seřazené podle popularity. Zůstaneme-li u hojně využívaných služeb společnosti Google, Ajax využívají i populární Google mapy, kdy se uživateli nahrává

nová poloha a nový snímek během procházení mapy bez nutnosti načtení celé stránky. [8] [9]



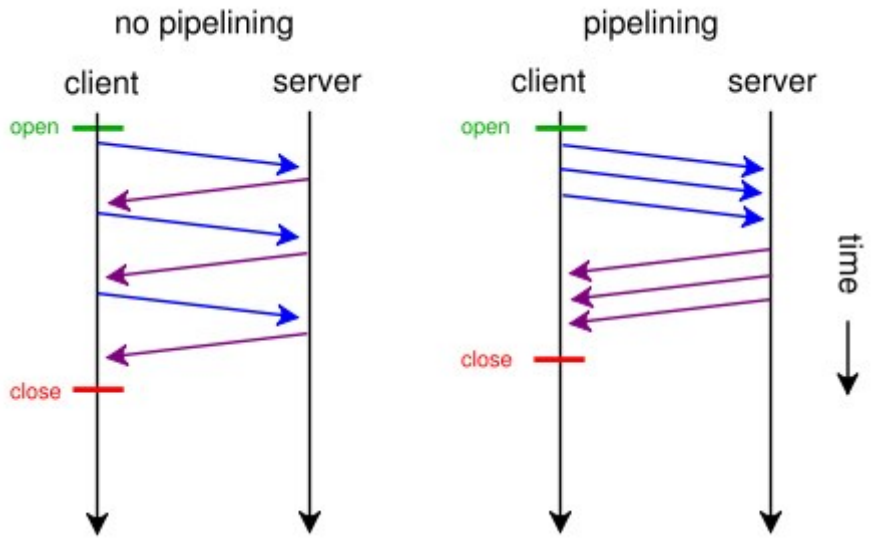
Obrázek 2.3: Použití Ajaxu v Google Našeptávači



Obrázek 2.3: Použití Ajaxu při vykreslování Google map

Kromě technologií LINQ a Ajax se verze 3.5 dostalo i několik jiných aktualizací. Přibyly komponenty pro manipulaci s daty (*ListView*, *DataPager*), WCF podpora pro RSS feedy, JSON nebo POX formát dat. Nová verze ASP.NET nabízí také

podporu tzv. HTTP pipelining (zřetězení http požadavků). Toto zřetězení je velice užitečné, protože dovoluje aplikaci poslat více požadavků najednou skrz jedno TCP připojení, díky čemuž se významně snižuje čekání na odpověď ze serveru. Ne všechny prohlížeče ale mají zřetězení v základní konfiguraci povoleno kvůli kolizím se staršími servery. [3] [10]



Obrázek 2.4: Rozdíl mezi klasickým zpracováním požadavků (vlevo) a zřetězením (vpravo) [37]

2.3.4 ASP.NET 4.0

V dubnu roku 2010 se weboví vývojáři dočkali čtvrté verze ASP.NET, která se snaží přinést další zlepšení.

- Služby jádra, která zahrnují nové API umožňující rozšířit cache paměť a podporu komprese stavových dat v relaci.
- Webové formuláře nabízejí více integrovanou podporu ASP.NET směrování, rozšiřují podporu webových standardů, lepé pracují s komponentami pro ovládání dat a nabízejí nový způsob pro manipulaci s ViewState (stavová data můžeme uchovávat jen tam kde opravdu potřebujeme).
- Implementace Ajaxu přináší přídavnou podporu pro client-based Ajax aplikace.
- Nové nástroje pro automatizaci typických úkonů při nasazení aplikace. [11]

2.3.5 ASP.NET 4.5

S příchodem nového .NET Frameworku verze 4.5 ve druhé polovině roku 2012 se vývojáři webových aplikací dočkali i nového ASP.NET 4.5. Jak se již stává tradicí, s novou verzí frameworku firma Microsoft vydala i nové Visual Studio 2012. Novinky v ASP.NET 4.5:

- Podpora HTML5 formulářů
- Automatický převod uživatelských vstupů na datové typy při bindování (ve Web Forms).
- Podpora pro validační JavaScript na straně klienta.
- Integrované rutiny z knihovny AntiXSS (dříve externí), které chrání před útoky prostřednictvím skriptování mezi servery.
- Podpora protokolu WebSockets.
- Asynchronní čtení a zapisování HTTP požadavků a odpovědí. [12]

Zajímavou novinkou je podpora protokolu WebSockets. Jedná se o obdobné využití síťových socketů v prostředí webového vývoje, kde může webová stránka navázat obousměrné spojení se serverem, který WebSockets podporuje, a vyměňovat si informace v reálném čase. Tato technologie je využitelná především pro různé kooperativní služby nebo tvorbu on-line chatů. [39]

2.3.6 ASP.NET 4.5.1

Prozatím nejnovější verze ASP.NET pro webový vývoj byla ohlášena v říjnu roku 2013 spolu s novou verzí Visual Studia. Jak již číselné označení napovídá, nejedná se o plnohodnotnou novou verzi frameworku, ale o update verze 4.5. Nejdůležitější změny, které nová verze přináší, se týkají vylepšení funkcí pro vývojáře (*debugging*) a zlepšení výkonu aplikací. [33]

2.4 Vývojové modely

Shrnutí pojmu ASP.NET je takové, že se jedná se o vývojový rámec (*framework*) pro vytváření webových stránek s technologiemi HTML, CSS, JavaScript a skriptováním na

straně serveru. ASP.NET nabízí tři způsoby jak vytvořit webovou aplikaci. Nazývají se vývojové modely (*development models*) a jsou to Web Pages, Web Forms a MVC. [13]

2.4.1 ASP.NET Web Pages

Model Web Pages je ze všech třech zmiňovaných nejjednodušší. Je to způsob vytváření webových stránek, který je velice podobný programování jazykem PHP. K tvorbě se používá editor WebMatrix, ve kterém má programátor plnou kontrolu nad HTML kódem, kaskádovými styly a JavaScriptem. Programátoři webových stránek v PHP nebo klasickém ASP, vědí, že skripty se vkládají přímo do HTML kódu a cílový web se „staví“ stránku po stránce, které mezi sebou komunikují. Tímto způsobem pracuje i model Web Pages a proto je velmi vhodný pro méně zkušené vývojáře, kteří se s frameworkem ASP.NET teprve seznamují. [14]

2.4.2 ASP.NET Web Forms

Web Forms je nejstarší vývojový model, kdy se stránky píší kombinací HTML a serverového kódu. Tento kód je kompilovaný na serveru a generuje HTML kód, kterým jsou stránky prezentovány. Díky tomu tvorba webových stránek připomíná programování desktopových aplikací pod operačním systémem Windows. Při programování se používají ovládací prvky (*Controls*), které mají svoje vlastnosti a události. [15] [16]

Jak tedy Web Forms fungují? Podobně jako ve Windows Forms, které se používají pro vývoj desktopových aplikací na systému Windows, zde existují události, které se provádějí v určitém pořadí při běhu webového formuláře. Dále na stránce existují události, které reagují na činnost uživatele. Předpoklad, že server zpracuje ASP a HTML klasickým způsobem lineárně od shora dolů, je mylný. Ve Web Forms je tomu jinak. Stejně jako ve Windows Form, i ve Web Forms procházejí události cyklem, kdy načítají stránku (*Load*), poté ji zobrazí (*Draw*) a nakonec zavřou (*Unload*). Během tohoto procesu mohou být volány různé metody. Pokud je stránka od klientského prohlížeče vyžaduje, tak se načte knihovna DLL (*Dynamic-link library*), která obsahuje jak značky na ASPX stránce, tak zdrojový kód. V případě, že se upraví zdrojový kód nebo ASPX stránka, DLL knihovna, která ji zastupuje, se dynamicky obnoví při dalším

volání upravené stránky. Knihovna je uložena na disku pokaždé, když je stránka generována.

Cyklus Web Forms:

1. Událost *Init* nejprve nastaví stránku do počátečního stavu, jak je popsána značkami v ASPX souboru. Pokud je stránka posílána sama na sebe, událost také zkontroluje data, která by mohla být uložena ve *ViewState*. Zde je také možné událostí *Page_Init* upravovat počáteční stav stránky.

2. V dalším kroku se spustí událost *Load*. Tou si můžete ověřit, zda se stránka pouští poprvé nebo zda po návratu na ní uživatel použil nějaký ovládací prvek. Pokud se pouští stránka poprvé, zajišťuje načtení dat, které při dalším spuštění načítat nechceme.

3. Další krok se spustí pouze v případě, proběhne-li *post back* stránky. Jedná se o případ, kdy se stránka zobrazí znovu v aktualizované podobě (např.: uživatel smaže záznam v tabulce a stránka se načte již bez smazaného záznamu). V tomto případě se spustí kontrolní události, které zaznamenají změny a zobrazí stránku ve správné podobě.

4. Nyní přichází na řadu vykreslení stránky v prohlížeči. Pokud jsou nějaké informace ve *ViewState*, uloží se do skrytých polí v těle stránky. Díky tomu může ASP.NET při další *post back* události obnovit stránku do předchozího stavu.

5. Na konec celého cyklu přichází událost *Page_Unload*. Tato událost se stará o šetrné ukončení kódu, který právě probíhal. Pod tím se myslí vyčištění logovacích úloh nebo vymazání stránky z paměti serveru. [17]

2.4.3 ASP.NET MVC

Než bude představen třetí vývojový model, kterým je ASP.NET MVC, bude uvedeno několik věcí, které nejsou ve Web Forms úplně dobré:

- Velikost *ViewState* – data v tomto mechanismu mohou nabýt velikosti stovek kilobajtů. Ty pak putují tam a zpět s každým požadavkem a zpomalují rychlost reakce na uživatelský požadavek.
- Životní cyklus stránky – mechanismus reakcí na události, který spravuje serverový kód, může být velice komplikovaný a náchylný na chyby.
- Falešný dojem oddělení aplikační logiky a prezentační vrstvy – vývojáři jsou někdy nuceni míchat serverový a prezentační kód. Například při manipulaci s databázovými daty.
- Omezená kontrola nad HTML – komponenty generují HTML samy a to ne vždy může mít takovou podobu, kterou byste očekávali.
- Nízká testovatelnost – v době vývoje Web Forms její designéři nepředpokládali, že se automatické testy stanou součástí webového vývoje. [22]

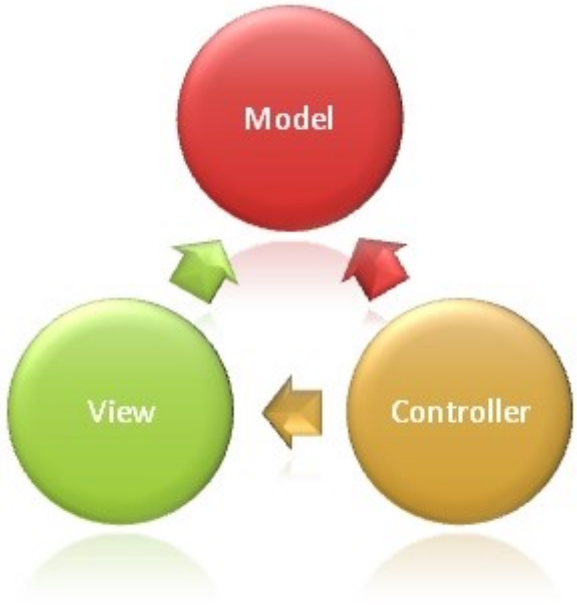
Nyní následuje hlavní obsah této podkapitoly. ASP.NET MVC je implementací populárního návrhového vzoru *Model-View-Controller*. V této architektuře je aplikace tvořena třemi logickými vrstvami, které lze samostatně upravovat. Architektura zajišťuje, aby dopad úpravy jedné vrstvy byl na ty ostatní co nejmenší. Pokud to je vyloženo tímto způsobem, není nic jednoduššího než si otevřít Visual Studio a začít programovat svoji první ASP.NET MVC aplikaci. Není to ale tak jednoduché, jak se na první pohled může zdát. Při samotném vývoji se musí myslet na to, co kam patří a musí se řešit konkrétní otázky.

Nejprve budou představeny jednotlivé části MVC architektury:

1. Model – tzv. business vrstva, která pracuje s daty v aplikaci. Objekty modelu typicky načítají data z databáze a ukládají do ní.

2. View – zobrazuje data, která dodává business vrstva.

3. Controller – tato vrstva reaguje na požadavky uživatelů a svazuje předchozí dvě vrstvy. Tato vrstva čte data z View, ovládá data vložená uživateli a předává je vrstvě Model.

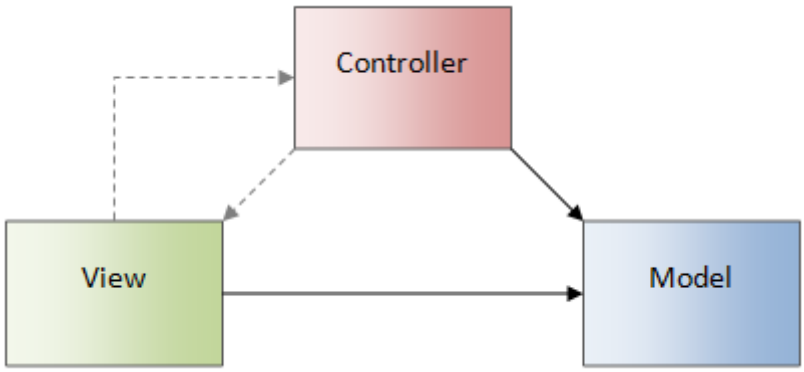


Obrázek 2.5: Tři vrstvy, ze kterých se skládá MVC architektura [18]

Před započetím vývoj se tedy musí zařadit různé činnosti aplikaci do správné vrstvy. Vyvstávají otázky, zda patří funkčnost aplikace do Modelu nebo Controlleru? Nebo zda má mít View přímou vazbu na Model? Tyto otázky nelze jednoznačně zodpovědět, ale je to přesně ten typ problémů, které při vývoji bude programátor řešit. [18] [20]

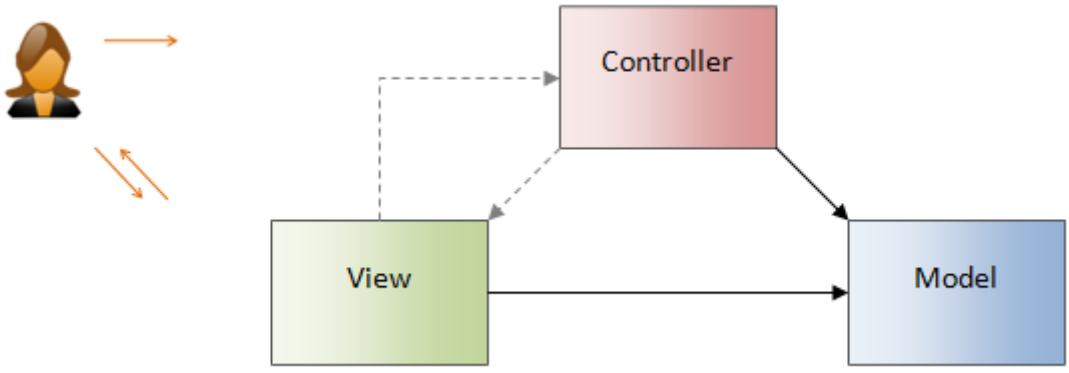
Architektura MVC je poměrně mladá záležitost, která se v posledních letech stává stále populárnější. Může to být způsobeno tím, že různé technologie mají ranou fázi vývoje za sebou a kromě základní výbavy se snaží nabídnout kvalitní architekturu. Proto například firma *Zend Technologies* investuje nejen do PHP, ale i do jejího *Zend Frameworku*. Při snaze nabídnout kvalitní architekturu tak Microsoft na konci roku 2007 vydal ASP.NET MVC, který se v roce 2013 dočkal již své páté verze.

Základní myšlenku MVC, která je nastíněna v úvodu této podkapitoly, pochopí rychle každý. Ale proniknutí do všech detailů je běh na dlouhou trať. Zásadní věcí pro pochopení MVC architektury je také oddělovat celek do dvou částí. První částí je MVC jako obecná architektura a přístup k tvorbě aplikací. A druhou částí jsou jednotlivé variace MVC. Ne zrovna pozitivním zjištěním je, že variací je několik a to pravé umění je v jejich ovládnutí. Velmi pozitivní věcí ale je, že právě různé variace MVC architektury dokáží vývojáři zodpovědět konkrétní otázky, které byly zmíněny dříve v této podkapitole. [20]



Obrázek 2.6: Vazby mezi vrstvami MVC [20]

Obrázek 2.6 ilustruje vazby mezi jednotlivými částmi MVC architektury. Existují zde pouze dvě přímé vazby. *Controller* má přímou vazbu na *Model* kvůli úpravě dat a *View* má přímou vazbu také na *Model*, aby mohl data zobrazovat. Žádné další přímé vazby nejsou podstatné. V závislosti na konkrétní variaci MVC poměrně často existuje vazba mezi *View* a *Controllerem*. Vazby, které jsou vyznačené v obrázku přerušovaně, mohou být buď obousměrné, nebo jednosměrné. Naopak nikdy neexistuje přímá vazba z *Modelu* na ostatní dvě vrstvy. V některých schématech se můžete setkat s nepřímou vazbou z *Modelu* na *View*. Touto nepřímou vazbou se myslí, že vrstva *View* je informována o nějaké změně dat v *Modelu*. V každém případě ale platí, že přímá vazba z Modelu na některou ze zbývajících vrstev je hrubou chybou v návrhu aplikace.



Obrázek 2.7: Komunikace s uživatelem [20]

Na obrázku 2.7 je znázorněna komunikace s uživatelem. Výstup k uživateli má vždy na starosti vrstva *View* (zobrazení webové stránky). Při zpracování uživatelského vstupu ale existují dvě odlišné možnosti:

1. Ve „widgetových“ systémech (Java Swing, Windows Forms, WPF, Silverlight, Flex, ASP.NET Web Forms) uživatelský vstup ošetřují komponenty samy. Příkladem může být tlačítko, které umí reagovat na událost *Click*. V tomto případě uživatelský vstup zpracovává vrstva *View*.
2. V systémech, kde komponenty neexistují je potřeba provádět ošetření uživatelského vstupu na jiném místě. V tomto případě se o vstup stará *Controller*. [20]

Architektura MVC se používá pro aplikace všeho druhu. To znamená, že je možné ji použít v desktopových, mobilních i webových aplikacích. Co přesně ale jednotlivé písmena M, V a C představují při webovém vývoji? *Model* je nejjednodušší, protože je identický ve všech typech aplikací. Obsahuje data a logiku a proto nemá s prezentací nic společného.

Bez rozmyslu je možné jako *View* označit HTML kód, protože prezentuje aplikaci. Tato úvaha je chybná, protože v tom případě je možné jako *View* v desktopových aplikacích označit pixely na obrazovce. Toto jsou pouze nízkoúrovňové vykreslovací instrukce, ale ve skutečnosti je *View* ve webovém prostředí serverový kód, který se stará o generování HTML. Vrstvou *View* je tedy, například v prostředí ASP.NET, šablonovací jazyk ASPX nebo Razor, které budou více popsány v pozdější části této práci.

Controller ve webovém prostředí většinou není jedna ucelená část. První částí je tzv. *Front Controller*, který odchytává všechny HTTP požadavky. Druhou částí je pak konkrétní *Controller*, který ony požadavky obdrží přeposlané od *Front Controlleru*. Konkrétní *Controller* obdržená data uloží do *Modelu* a prováže ho s konkrétním *View*, který vyrenderuje požadovaný výstup. Výstupem nemusí být nutně HTML, ale také třeba data ve formátu XML nebo JSON. [21]

3 Požadavky na aplikaci

Tato kapitola začíná praktickou část diplomové práce. V následujících třech kapitolách bude demonstrován vývoj webové aplikace technologií ASP.NET. Webová aplikace bude mít podobu sportovního portálu. Není ale potřeba tvořit web, který bude zahrnovat všechna sportovní odvětví a tak se bude portál točit pouze okolo fotbalu, který autor této práce preferuje.

Nebude se začínat tvořit webová aplikace jen tak z ničeho. Nejprve se stanoví požadavky, které budou rozděleny na funkční a technické. Funkční požadavky se budou zaměřovat na to, co bude webový portál umět. Do této kategorie patří prakticky veškeré funkce, které je schopný nabídnout čtenářům, registrovaným uživatelům, redaktorům a administrátorům. Oproti tomu technologické požadavky stanovují, jak to bude portál umět. Sem budou patřit veškeré technologie, které budou pro tvorbu portálu použité.

Pro získání přehledu o tom, které rysy jsou pro fotbalový portál žádoucí, se zmapuje situace na českém internetu. Nyní přijdou na řadu malé analýzy vybraných českých portálů, na kterých bude ukázáno, co na nich autor této práce oceňuje a co mu tam naopak chybí a považuje to buď za důležité, nebo jako součást uživatelského pohodlí.

3.1 České portály

Na českém internetu je stránek věnujících se fotbalové tématice velké množství. Vybrat několik portálu pro ukázku špatných a dobrých věcí není jednoduché. Do výběru nebude zahrnuta například fotbalovou sekci na zpravodajském serveru iDnes, protože je to součást mnohem většího celku. Podle vyhledávání Googlu přes klíčové slovo „fotbal“ a bližším prozkoumání jednotlivých portálů jsou ve výběru tři weby.

3.1.1 FotbalPortal.cz

První portál, který bude ukázán a uvede problematiky, je FotbalPortal.cz (dostupný na *http://www.fotbalportal.cz*).

Obrázek 3.1: Úvodní strana webu FotbalPortal.cz

Na obrázku výše je otisknuta titulní strana tohoto fotbalového portálu. Celkový dojem jinak příjemné grafiky bohužel kazí výrazné reklamy, především na sázkovou kancelář. Po bližším zkoumání tohoto portálu, je zřejmé, že obsahově je na tom velice dobře. Každý den redaktoři píší nové články. Výrazným nedostatkem z hlediska obsahu je absence výsledkového servisu z drtivé většiny evropských a světových soutěží. Portál se kromě našeho českého fotbalu zaměřuje pouze na Anglii, Španělsko, Německo a Itálii. K tomu přidává servis z evropských pohárových soutěží a některých reprezentačních akcí.

Přihlášený uživatel má možnost měnit barvené téma stránky, přispívat pod články a zúčastnit se tipovací soutěže, kterou portál provozuje. V úpravě svého profilu si kromě osobních údajů může uživatel nastavit oblíbený fotbalový klub a oblíbeného

fotbalistu. Portál se snaží své registrované uživatele sbližovat tím, že jim umožňuje uzavírat přátelství s jinými uživateli a zasílat mezi sebou soukromé zprávy.

Na konci krátkého představení budou zhodnoceny klady a zápory portálu. Mezi pozitivní rysy se řadí dostatečná aktualizace obsahu, možnost účasti na tipovací soutěži pro registrované uživatele a také sociální pojetí portálu, kdy si uživatelé tvoří seznamy přátel a ve svém profilu prezentují svoje fanouškovské preference.

Do negativních prvků portálu patří velice omezený výsledkový servis, kdy se čtenář dozvídá aktuální výsledky a pořadí v tabulkách pouze z malého množství evropských zemí. Trochu rozptylujícím až odpuzujícím prvkem může být také velká reklamní plocha na portálu. Propagace v dnešním internetovém světě sice hraje velkou roli, ale někdy to může být vzhledem ke koncovému čtenáři portálu na škodu.

3.1.2 eFotbal.cz

Dalším portálem, který bude představen je eFotbal.cz (dostupný na *http://www.efotbal.cz*). Tento portál má mnoho společných rysů s předešlým, ale hlavně reklamní plochy řeší způsobem, který je příznivější pro čtenáře. Grafické rozhraní je poutavé a nevtíravé.

Po obsahové části portál nabízí velké množství aktuálních článků z různých evropských i světových soutěží. S prvním představeným portálem má ale společné to, že podrobné tabulky poskytuje pouze pro českou soutěž a několik nejdůležitějších evropských. Pro tyto soutěže také nabízí databázi klubů a hráčů včetně informací a statistik. Součástí portálu je stránka s „živými výsledky“, kde jsou živě spravované výsledky velkého množství evropských, mezinárodních i světových soutěží a také výsledky z těchto soutěží týden zpět a rozpis zápasů týden dopředu. Nejedná se však o práci portálu eFotbal.cz, ale o vestavěnou službu, kterou poskytuje webová aplikace Livescore (dostupná na *http://www.livescore.in*). Velkou devizou portálu je podrobné zpravodajství z regionálního fotbalu v rámci České Republiky. Portál se věnuje kromě nejvyšší české soutěže i těm nižším od celorepublikových až po nejvyšší krajské. Pro každý kraj je zpracována sekce, ve které jsou aktuální zprávy a výsledky z krajských přeborů.

Obrázek 3.2: Domovská stránka webu eFotbal.cz

Přihlášenému uživateli eFotbal.cz nabízí standardní možnosti úpravy svého profilu a komentování článků. Oproti předchozímu portálu nenabízí možnost navazování virtuálních přátelství, ale registrovaný uživatel se může účastnit konverzací v obecném diskuzním fóru. Registrovaný uživatel se může stejně jako na FotbalPortal.cz účastnit tipovací soutěže.

Tento web je zajímavý především svým zaměřením na české nižší soutěže, ale chybí mu lepší přehled z ostatních evropských a světových. V celkovém hodnocení převažují kladné stránky. Mezi ně patří tipovací soutěž pro registrované uživatele nebo velký přehled živých výsledků. Portál se také liší tím, že obsahuje malé diskuzní fórum, které není rozčleněno na kategorie, ale nabízí diskuzi na témata, která vloží registrovaní uživatelé.

3.1.3 EuroFotbal.cz

Poslední fotbalovým webem, který bude představen je EuroFotbal.cz (dostupný na *http://www.eurofotbal.cz*). Hned na začátku je patřičné uvést, že tento portál funkčně i obsahově převyšuje předchozí dva weby a řadí se tedy na pomyslné první místo.

Jako první bude rozebrána obsahová část, která je na vysoké úrovni a mezi českými fotbalovými portály nemá konkurenci. Tým redaktorů vydává zajímavé články každý den a dává prostor i svým čtenářům, kteří mají možnost svoje názory sepsat do článků nebo fejetonů. Během fotbalové sezóny se na stránce objevují souhrny zápasů nejznámějších fotbalových soutěží, kde nechybí sestavy týmů, statistiky a podrobná reportáž.

Obrázek 3.3: Úvodní stránka webu EuroFotbal.cz

Kromě reportáží z nejznámějších lig se na webu nachází databáze soutěží, ve které si čtenář může najít výsledky a tabulky prakticky z jakékoliv evropské profesionální soutěže. Fascinujícím příkladem může být skutečnost, že aktualizované jsou i tabulky a výsledky z tak malých nebo vzdálených zemí jako jsou Kazachstán, Andorra nebo Lucembursko. Databáze obsahuje kromě velkého množství soutěží i ještě větší množství profesionálních klubů a hráčů hrajících po celém světě. V případě evropských klubů a hráčů jsou zde k dispozici kompletní statistiky i několik let dozadu. V případě zájmu si tedy není problém najít konkrétního hráče a zjistit, kde za poslední léta působil a s jakými úspěchy.

Kromě neskutečně obsáhlého výsledkového servisu, ale tento portál nabízí i další prvky internetové zábavy. Stejně jako předchozí portály nabízí tipovací soutěž nebo on-line výsledky převzaté z aplikace Livescore.

Grafické provedení webu je velice zdařilé. Navigace, články, krátké zprávy a výsledky jsou přehledně strukturované a do designu nezasahují ani reklamní bannery, které jsou do webu vhodně rozmístěné.

Vedoucí místo mezi českými portály je podpořeno i nevyvratitelnou statistikou. Bude řeč o registrovaných uživatelích. Jelikož nám není známé přesné číslo registrovaných uživatelů na těchto třech portálech, které jsou popisovány, bude se vycházet z počtu komentářů pod zveřejněnými články. Zatímco se na portálech *FotbalPortal.cz* a *eFotbal.cz* počet komentářů pohybuje maximálně v řádech několika desítek komentářů, *EuroFotbal.cz* zaznamenává pod články rozsáhlé diskuze uživatelů, které čítají i několik stovek příspěvků. Pokud se jedná o články, které se věnují těm nejznámějším fotbalistům a týmům, počet komentářů se leckdy přežene hodně přes tisícovku.

Přihlášeným uživatelům portál nabízí běžné možnosti úpravy účtu, komentování článků a účasti v tipovacích soutěží. Další funkcí, kterou mohou uživatelé využít, je nahrávání oblíbených videí, které se agregují na samostatné podstránce. V sekci on-line výsledků běží také chat pro přihlášené uživatele.

Web *EuroFotbal.cz* je možno hodnotit jen a pouze kladně. Pokud se přičte ještě skutečnost, že správci webu dávají běžným čtenářům možnost psát články, ve kterých poodhalují svoje názory na konkrétní dění ve fotbalovém prostředí, vychází nám web, který funguje na špičkové úrovni. Na nejvyšší místo v pomyslné pyramidě pozitiv se určitě řadí obsahová část webu, která nemá mezi českými portály moc velkou konkurenci.

3.2 Funkční požadavky

V této podkapitole budou stanovené funkční požadavky webového projektu. Jde o ty činnosti, které chce aplikace nabídnout uživatelům. Pro třídění uživatelů se používá systém rolí, ve kterém má každý uživatel přidělenou jednu nebo i více rolí a podle toho mu aplikace zpřístupní konkrétní činnosti.

První část funkcionalit se týká úpravy uživatelského účtu:

- Vytvoření uživatelského účtu při registraci
- Změna hesla a kontaktních údajů

Další soubor funkcí patří k článkům na webu, videopříspěvkům a jejich komentářům. Zde budeme chtít delegovat tyto funkcionality:

- Vytváření, úprava a mazání článků
- Komentování článků
- Mazání a cenzurování komentářů v případě příspěvků, které se neslučují s podmínkami užívání uživatelského účtu
- Schvalování a časování vydání článků
- Vkládání a komentování videopříspěvků
- Schvalování a mazání videopříspěvků

Speciální sekcí na webu jsou „Názory“. Předlohou pro tuto část webu slouží známé sociální sítě Facebook a Twitter. Uživatelé zde dostanou možnost vyjádřit svůj názor na dění ve fotbalovém prostředí trochu jiným způsobem, než je na sportovních portálech běžné. Nabízené funkce budou:

- Vkládání a mazání názorů
- Navazování přátelství s jinými uživateli

Čtenáři webu budou mít také možnost hlásit se do fanklubů týmů. Nebudou je sami vytvářet, aby se předešlo duplicitám, které známe v případě tvoření skupin na sociální síti Facebook, ale budou se hlásit přímo v profilu klubu. U každého klubu tedy bude možné vidět, kolik čtenářů mu fandí a web může lehce generovat žebříčky nejoblíbenějších. Jeden čtenář se může hlásit do více fanklubů. Nabízené funkce:

- Přidání oblíbeného klubu do svého seznamu
- Odebrání oblíbeného klubu ze svého seznamu

Web bude také svým čtenářům nabízet statistiky z odehraných zápasů, tabulky a přehledy hráčských kariér. Plánované funkce:

- Zapisování zápasových výsledků
- Zapisování statistik do hráčských kariér

Poslední skupina funkcí se stará o uživatelské účty na webu a hlídá slušné meze příspěvků v komentářích, názorech a videopříspěvcích:

- Schvalování a mazání videopříspěvků
- Mazání neslušných komentářů a názorů
- Správa uživatelských účtů (změna údajů, deaktivace)

Pro rozdělení výše jmenovaných funkcí bude vytvořeno 5 rolí, které budou přidělovány uživatelům:

1. Běžný uživatel je základní role, ze které vycházejí všechny níže zmíněné. Běžným uživatelem se rozumí registrovaný návštěvník webu. Tento uživatel má možnost komentovat zveřejněné články, upravovat svůj uživatelský profil, posílat videopříspěvky, psát své názory, navazovat přátelství s jinými uživateli a zapisovat se do fanklubů.
2. Redaktor je první z rolí, které se starají o obsah webu. Tato role oproti běžnému uživateli nabízí psaní a úpravu článků.

3. Šéfredaktor je role, která dohlíží na zveřejňování článků. Má možnost upravit kterýkoliv článek a v prvé řadě schvaluje články napsané redaktory. Článek může nejen schválit, ale i určit čas vydání.
4. Role Editor vychází stejně jako předchozí z běžného uživatele. Rozdíl je v tom, že se nestará o články, ale o videopříspěvky a statistickou část webu. Editor schvaluje videopříspěvky zasílané ostatními uživateli, zadává výsledky zápasů a spravuje osobní statistiky fotbalistů.
5. Nejvyšší rolí je Administrátor. Jedná se o správce celého webu, který má v repertoáru všechny výše zmíněné činnosti. Mimo nich ještě spravuje diskuze a uživatelské účty. Tyto činnosti zahrnují hlavně cenzuru nevhodných příspěvků a následnou deaktivaci účtů problémovým uživatelům.

3.3 Technologické požadavky

Webová aplikace bude využívat verzi ASP.NET 4.5 a bude postavena na návrhovém vzoru MVC. Tento vzor jsme již popisovali v dřívější části práce. Aktuální verze v době, kdy byly započaty práce na projektu, je ASP.NET MVC 4.

Pro práci s databází bude aplikace používat ORM framework. Zkratka ORM znamená *object-relational mapping*. Těchto frameworků existuje celá řada a slouží k tomu, aby vývojář webové aplikace byl co nejvíce odstíněn od databáze na té nejnižší struktury ve smyslu psaní SQL dotazů a ručního vytváření tabulek. Pro účely ukládání a načítání dat z databáze se vytvoří třídy, které představují tabulky v databázi. V MVC architektuře tyto třídy zastupují část Model. ORM framework sám zařídí vytvoření a aktualizaci tabulek v databázi. Vývojář tedy vůbec nemusí ručně vkládat nebo upravovat tabulky. V případě ASP.NET použijeme ADO.NET Entity Framework (či pouze Entity Framework).

Výsledné webové stránky budou generovány v nejnovější specifikaci organizace W3C na strukturu webových stránek HTML5. Tato nová specifikace postupně nahrazuje starší HTML4 a XHTML. Snaží se zjednodušovat strukturu webu a nabízet vývojářům nové možnosti. HTML5 přináší vývojářům například snadnější vytváření oddílů webu, vkládání multimediálního obsahu nebo kreslící element *canvas* pro tvorbu vektorové grafiky a vkládání obrázků. [23]

V současné době jde téměř ruku v ruce s HTML5 také nová verze jazyka pro zobrazení elementů na webových stránkách CSS3. CSS3 oproti předchozím verzím nabízí vývojářům další zajímavé a estetické možnosti stylování obsahu. Některé nové rysy byly předtím nedosažitelné, jiných se dosahovalo velmi obtížně a to za pomocí různých javascriptových knihoven, nebo vkládáním obrázků, kde byl zamýšlený efekt již nakreslený. Jedná se například o oblé rohy elementů, stínování nebo využití dvojitého pozadí (jeden prvek může mít například dva různé obrázky na pozadí). Další novinkou je využití libovolného fontu. Webdesigner se už nemusí spoléhat pouze na úzký okruh fontů, které nabízí základní výbava internetového prohlížeče, ale může do webu zakomponovat prakticky jakýkoliv font, který tam pak kodér nadefinuje pomocí odkazu na soubor, kde je písmo uložené. CSS3 také nabízí 2D a 3D animace objektů, ale toto má zatím omezenou podporu prohlížečů. [24]

4 Návrh aplikace

Po specifikaci požadavků na webovou aplikaci přichází další krok vývoje – návrh aplikace. Základem většiny webových aplikací není z pohledu programátora uživatelské rozhraní, ale databázová vrstva. Hned po datech následuje logika aplikace, která s nimi pracuje. Až nad tímto je uživatelské rozhraní, které je prací spíše pro grafiky a webdesignery. Tento koncept není ničím novým. Je rozebírán už v jedné z dřívějších kapitol, kde byl popsán návrhový vzor MVC. Data jsou důvodem, proč návštěvník zavítá na většinu webů a u sportovního portálu tomu nebude jinak. Tato kapitola se bude věnovat diagramům, které se snaží co nejlépe popsat vyvíjenou aplikaci.

4.1 Diagram případů užití

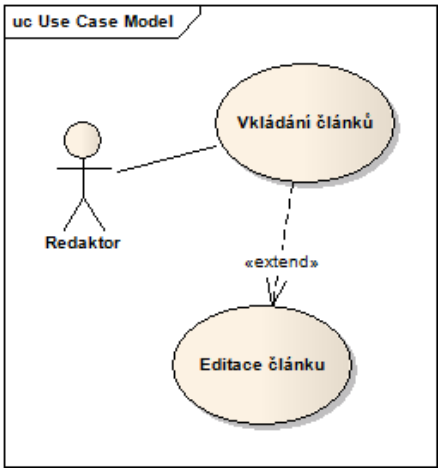
Tato kapitola se dostává k samotnému modelování naší webové aplikace. Bude popsána spíše teorie modelování, jejímž výstupem bude reálný diagram případů užití. Modelování případů užití je doplňkovým způsobem získávání požadavků. Funkční požadavky na systém byly nastoleny již v jedné z předchozích kapitol a nyní budou pouze převedeny jazykem UML do vizuální podoby.

Modelování se skládá z následujících aktivit:

- Nalezení hranic systému
- Nalezení aktérů
- Specifikace případů užití
- Určení scénářů
- Postupy se opakují, dokud nedojde k ustálení případů užití, hranic systémů a aktérů

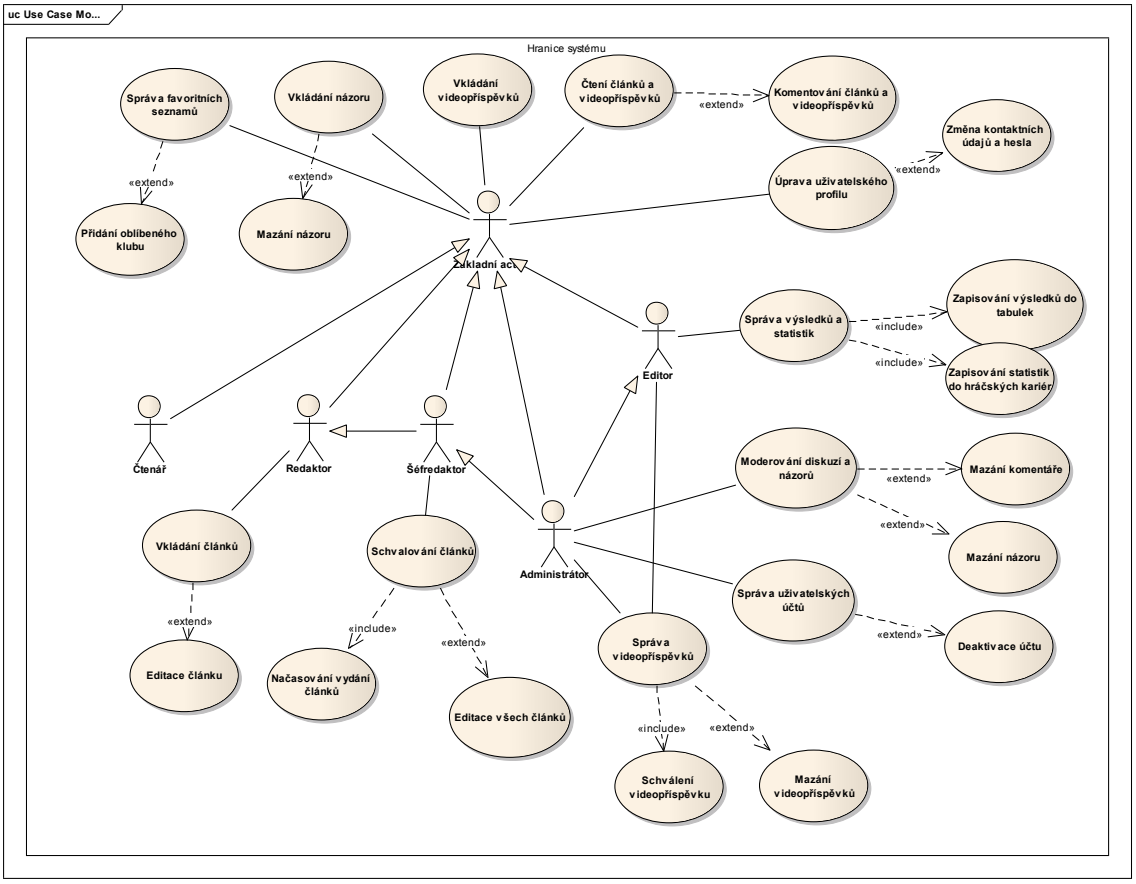
Hranici systému, které je ohraničení zobrazené kolem případů užití, se v UML 2 říká *subjekt*. Vytyčení hranice zdánlivě vypadá jako samozřejmá věc, ale i přesto se v některých případech může stát, že neúplné stanovení požadavků může způsobit krach projektu. Aktéři jsou role přidělené osobám (nebo i předmětům) používajícím systém. Případy užití jsou samotné činnosti, které mohou aktéři provádět. Aktér není jen osoba, která provádí činnost, kterou systém poskytuje, ale i třeba pracovník, který se o systém

stará a spravuje ho. Důležitou roli v některých softwarových aplikacích může hrát i čas, který může vystupovat jako aktér, v případě že se v určité časové úseky provádějí nějaké automatické činnosti. Vztahy mezi aktéry a případy užití se nazývají *relace*. [25]



Obrázek 4.1: Případy užití v případu redaktora

Případy užití musí být vždy iniciovány aktérem a jsou definované z pohledu aktéra. Ukázka malého diagramu případů užití je na obrázku č. 4.1. V tomto případě je znázorněn aktér *Redaktor*, který má případ užití *Vkládání článků*. Tento případ užití je vazbou *extend* rozšířen o další případ užití *Editace článku*.



Obrázek 4.2: Úplný model případů užití

Na obrázku 4.2 je znázorněn úplný model případů užití v aplikaci. Při bližším prozkoumání je patrné, že v souladu s podstatou případu užití, se jedná o model funkčních požadavků ze třetí kapitoly této práce. Tento diagram obsahuje několik vazeb mezi aktéry a případy užití, které budou vysvětleny:

- *Use* – základní vazba mezi aktéry a případy užití, která je znázorněna jednoduchou čárkou. Říká, že aktér může využít vykonat konkrétní případ užití.
- *Extends* – jedná se o vazbu mezi případy užití. Ty případy užití, které jsou touto vazbou vázané na jiné, aktér může vykonat, ale také nemusí. Tato vazba znázorňuje většinou nepovinné kroky v nějaké posloupnosti činností. Aktér může například vytvořit článek a pak ho může editovat.
- *Include* – tento vztah naopak určuje případy užití, které na sebe navazují a je nutné je spolu vykonat. V diagramu například šéfredaktor při schvalování článku musí načasovat jeho vydání.

- *Generalization* – generalizace je vztah mezi aktéry. V ukázkovém modelu je využíván především proto, aby do něj vnesl určitý pořádek. V tomto případě například všichni aktéři, kromě administrátora, vycházejí ze základního aktéra. Ten sdružuje všechny činnosti, které dědí ostatní aktéři. Každý další aktér v pyramidě přidává specifické vlastnosti pro svoji roli. [30]

Součástí rozsáhlejšího diagramu případu užití bývají i scénáře. V této práci je nijak rozebírat nebudeme, protože úkolem není sestavení příručky jazyka UML, ale přesto se o scénářích krátce zmíníme.

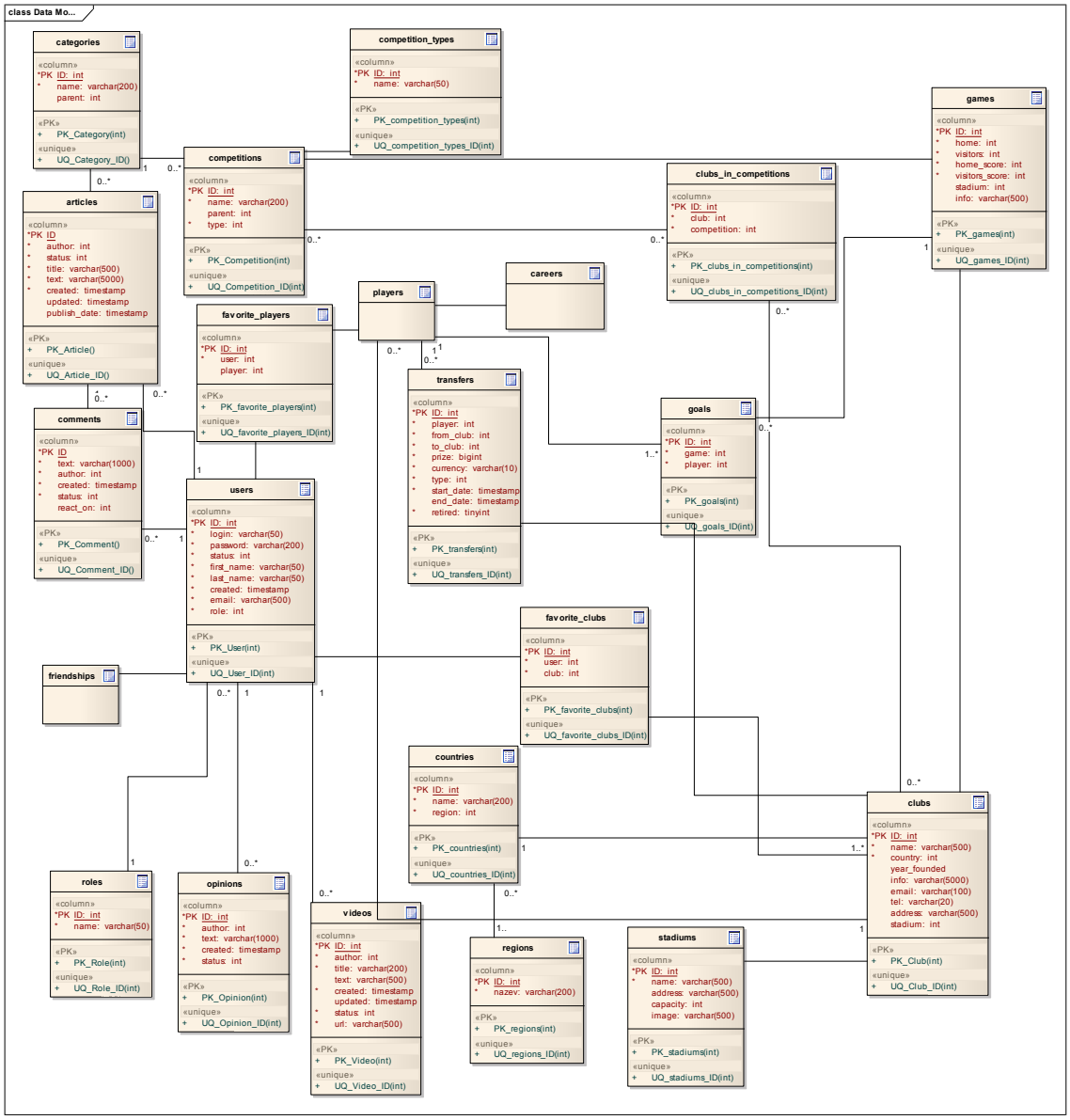
Pokud mají případy užití více kroků, jsou uvedeny v toku událostí. Toky jsou hlavní a alternativní. Hlavní je označován také jako *hlavní scénář* a alternativní toky jako *vedlejší scénáře*. V hlavním scénáři jsou kroky, které vždy probíhají dle očekávání. Pokud dochází k nějaké chybě, odchylce nebo přerušení toku, následující kroky popisuje vedlejší scénář. Scénáře jsou slovní popisy, které jasně vysvětlí kdo co, ve které chvíli provede určitou funkci. Scénáře se mohou také větvit a využívat podmínku (klíčové slovo *KDYŽ*), případně se mohou různé posloupnosti vícekrát za sebou opakovat ve *for* nebo *while* cyklu.

Alternativní scénáře ošetřují chyby, které mohou být způsobeny hlavním scénářem. Každý případ užití by měl mít vždy přesně jeden hlavní scénář, ale může mít mnoho alternativních. Jak určit, který počet je ideální? Ze strany architekta musí být snaha omezit jejich počet na nezbytné minimum. Docílení lze dosáhnout dvěma způsoby:

1. Vybráním nejdůležitějších vedlejších scénářů a jejich dokumentováním
2. Redukovat skupiny podobných vedlejších scénářů a doplnit je o upřesňující poznámky

Ve skutečnosti nelze zmapovat přesný počet vedlejších scénářů, proto je nutné vybrat ty nejdůležitější. Důležité je také mít na paměti, že případy užití se zachycují za účelem pochopení chování systému, ne pro vytvoření kompletního modelu případu užití. [25]

4.2 Entity-relationship model



Obrázek 4.3: Rozpracovaný E-R model

Nyní bude lehce přiblížen návrh databázové struktury projektu. Tato kapitola nebude rozebrána do větší hloubky, protože v projektu bude použit databázový framework, který do jisté míry řeší databázovou vrstvu za vývojáře. Databázový framework se nazývá Entity Framework a bude mu věnována jedna z nadcházejících podkapitol.

Na obrázku 4.3 je Entity-relationship model (dále jen ER model), který popisuje vztahy mezi entitami, které jsou v tomto případě tabulky v databázi. Tento diagram se skládá ze zástupců tří komponent:

1. Entita – v kontextu E-R modelu je entita zástupcem dat, které mají jasnou strukturu. Diagram obsahuje mimo jiné informace o tom, jak budou vypadat tabulky hráčů a uživatelů. Hráč a uživatel v tomto kontextu figurují jako dvě rozdílné entity.
2. Vztah – propojení mezi entitami. Každá entita má určitý vztah jen s několika dalšími entitami, v konečném důsledku by měly být všechny entity propojené a tvořit jeden celistvý systém.
3. Kardinalita – detailněji popisuje vztah mezi entitami. Každého vztahu se mohou obě entity účastnit pouze jednou (1:1), obě vícekrát (N:M) nebo jedna z entit pouze jednou a druhá entita několikrát (1:N). [25]

5 Implementace

Tato kapitola se dostává k implementaci projektu v souladu se stanoveným návrhem. Postupně projde všechny hlavní body implementace a vysvětlí různá úskalí při vývoji a budování webové aplikace. Nejprve bude představen nástroj, který bude použit pro psaní zdrojového kódu – Visual Studio. Po teoretickém úvodu a založení nového projektu se přijdou na řadu praktické ukázky vývoje. Bude ukázáno jak se držet struktury MVC, jak mapovat třídy modelu na databázi pomocí Entity Frameworku nebo jak aktualizovat databázovou strukturu. Stranou nezůstanou ani klasické úkoly pro webového vývojáře jako správa uživatelů nebo tvorba dynamických menu. Webová aplikace s fotbalovou tématikou si také žádá vyřešení evidence výsledků, stadionů, klubů a v neposlední řadě také hráčů a jejich statistik a kariér.

5.1 Visual Studio 2012

Jak již bylo výše zmíněno v této kapitole bude představen nástroj pro tvorbu webových aplikací na platformě .NET. Tím je balík nástrojů Visual Studio. První verze tohoto software se objevila již v roce 1995 a od té doby prošla dlouhým vývojem a přinesla mnoho zlepšení. Od té doby se Visual Studio dočkalo celkem devíti dalších verzí. Platformu .NET nicméně začalo Visual Studio podporovat až v době jejího vzniku, tedy ve verzi Visual Studio .NET z roku 2002. Od té doby každá nová verze Visual Studia koreluje i s novou verzí .NET Frameworku. [26]

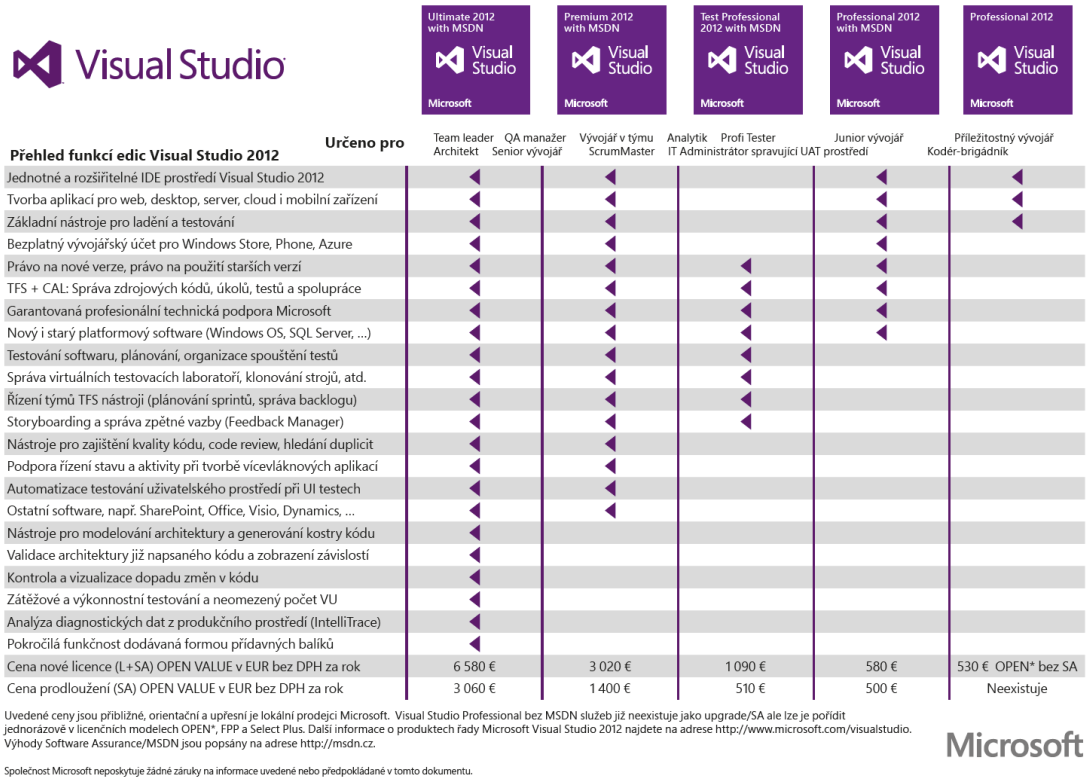
Obrázek 5.1: Logo nového Visual Studia [27]

Nejprve bude představena verze, která vyšla v roce 2012, nese označení Visual Studio 2012 a v jejímž prostředí byly zahájeny práce na projektu této diplomové práce. Ve skutečnosti to není jen jeden nástroj, ale balík nástrojů a služeb určený k vývoji softwarových aplikací pro desktopové i dotykové prostředí Windows, pro web s HTML5 standardem, mobilní zařízení, SharePoint i *cloud* prostředí. Visual Studio (dále jen VS) je k dispozici v několika edicích, které nabízejí různou škálu nástrojů a jsou

určeny pro odlišné skupiny pracovníků, protože ne každý využije všechny možnosti, které VS nabízí. Aktuální VS je k dispozici v pěti edicích:

1. *Professional* – nejnižší edice pro jednotlivce a programátory
2. *Test professional* – speciální edice pro testery
3. *Premium* – pro tvůrce komplexních aplikací
4. *Ultimate* – pro senior vývojáře, SW architekty, QA manažery a vedoucí týmů
5. *Team Foundation Server* – pro správu kódu a kompletní řízení vývoje [27]

Jak již bylo zmíněno, každá verze nabízí jinou škálu možností. Ty jsou uvedeny včetně cen za edici v následující tabulce.



Visual Studio	Ultimate 2012 with MSDN	Premium 2012 with MSDN	Test Professional 2012 with MSDN	Professional 2012 with MSDN	Professional 2012
Přehled funkcí edic Visual Studio 2012	**Určeno pro** Team leader Architekt, QA manažer Senior vývojář	Vývojář v týmu ScrumMaster	Analytik, Profi Tester IT Administrátor spravující UAT prostředí	Junior vývojář	Příležitostný vývojář Kodér-brigádník
Jednotné a rozšiřitelné IDE prostředí Visual Studio 2012	◀	◀		◀	◀
Tvorba aplikací pro web, desktop, server, cloud i mobilní zařízení	◀	◀		◀	◀
Základní nástroje pro ladění a testování	◀	◀		◀	◀
Bezplatný vývojářský účet pro Windows Store, Phone, Azure	◀	◀		◀	
Právo na nové verze, právo na použití starších verzí	◀	◀	◀	◀	
TFS + CAL: Správa zdrojových kódů, úkolů, testů a spolupráce	◀	◀	◀	◀	
Garantovaná profesionální technická podpora Microsoft	◀	◀	◀	◀	
Nový i starý platformový software (Windows OS, SQL Server, ...)	◀	◀	◀	◀	
Testování softwaru, plánování, organizace spouštění testů	◀	◀	◀		
Správa virtuálních testovacích laboratoří, klonování strojů, atd.	◀	◀	◀		
Řízení týmů TFS nástroji (plánování sprintů, správa backlogu)	◀	◀	◀		
Storyboarding a správa zpětné vazby (Feedback Manager)	◀	◀	◀		
Nástroje pro zajištění kvality kódu, code review, hledání duplicit	◀	◀			
Podpora řízení stavu a aktivity při tvorbě vícevláknových aplikací	◀	◀			
Automatizace testování uživatelského prostředí při UI testech	◀	◀			
Ostatní software, např. SharePoint, Office, Visio, Dynamics, ...	◀	◀			
Nástroje pro modelování architektury a generování kostry kódu	◀				
Validace architektury již napsaného kódu a zobrazení závislostí	◀				
Kontrola a vizualizace dopadu změn v kódu	◀				
Zátěžové a výkonnostní testování a neomezený počet VU	◀				
Analýza diagnostických dat z produkčního prostředí (IntelliTrace)	◀				
Pokročilá funkčnost dodávaná formou přídavných balíků	◀				
Cena nové licence (L+SA) OPEN VALUE v EUR bez DPH za rok	6 580 €	3 020 €	1 090 €	580 €	530 € OPEN* bez SA
Cena prodloužení (SA) OPEN VALUE v EUR bez DPH za rok	3 060 €	1 400 €	510 €	500 €	Neexistuje

Uvedené ceny jsou přibližné, orientační a upřesní je lokální prodejci Microsoft. Visual Studio Professional bez MSDN služeb již neexistuje jako upgrade/SA ale lze je pořídit jednorázově v licenčních modelech OPEN*, FPP a Select Plus. Další informace o produktech řady Microsoft Visual Studio 2012 najdete na adrese http://www.microsoft.com/visualstudio. Výhody Software Assurance/MSDN jsou popsány na adrese http://msdn.cz.

Společnost Microsoft neposkytuje žádné záruky na informace uvedené nebo předpokládané v tomto dokumentu.

Microsoft

Obrázek 5.2: Tabulka funkcí Visual Studio 2012 [27]

Společnost Microsoft také zvýhodňuje studenty, kterým nabízí některé produkty zcela zdarma a VS mezi ně patří. Zdarma je ale také tzv. Express edice, kterou může použít kdokoliv. Právě edice Visual Studio 2012 Express byla používána v této práci při tvorbě webové aplikace. Verze pro webový vývoj nabízí vše potřebné pro tvorbu

webové aplikace a zbytečně nezahlcuje programátora dalšími nástroji, které nepotřebuje.

5.2 Visual Studio 2013

V říjnu roku 2013 vydala společnost Microsoft novou verzi svého vývojářského balíku. S novou verzí přichází také nové funkce a vylepšení. Nová verze přináší lepší kompatibilitu s Windows 8, vývoj aplikací pro Windows Store, Windows Phone 8.1, nové ladící, diagnostické nástroje, lepší monitoring webových aplikací a samozřejmě také úpravu grafického rozhraní. [34]

Protože je Visual Studio 2013 nový a moderní nástroj pro programátory a vývojáře, musí i autor této práce reagovat na posun v technologiích webového vývoje. V průběhu prací se tedy hlavní vývojový nástroj změnil na Visual Studio 2013 Express pro webový vývoj.

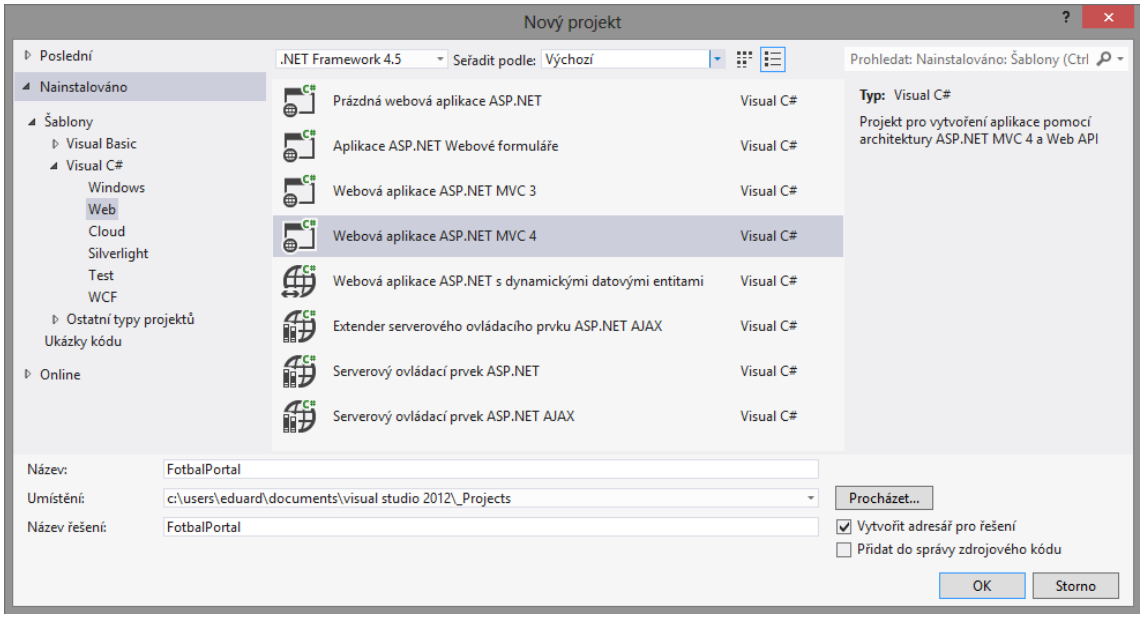
5.3 Visual Studio Online

S novou verzí Visual Studia se dostal vývojářům do rukou také nový nástroj Visual Studio Online. Nejedná se o prostředí k programování, ale dává si za cíl pokrývat vše ostatní co menší nebo větší týmy vývojářů a jejich vedoucích potřebují. Je to tedy online nástroj pro spolupráci, který poskytuje vlastní systém pro správu verzí založený na Gitu a spoustu dalších nástrojů, které umožňují řídit a plánovat vývoj projektů. Visual Studio Online také pomáhá s finálním nasazením a testováním. [35]

Tato práce není zaměřená na týmovou spolupráci vývojářů, proto se tomuto nástroji více věnovat nebude.

5.4 Nový projekt

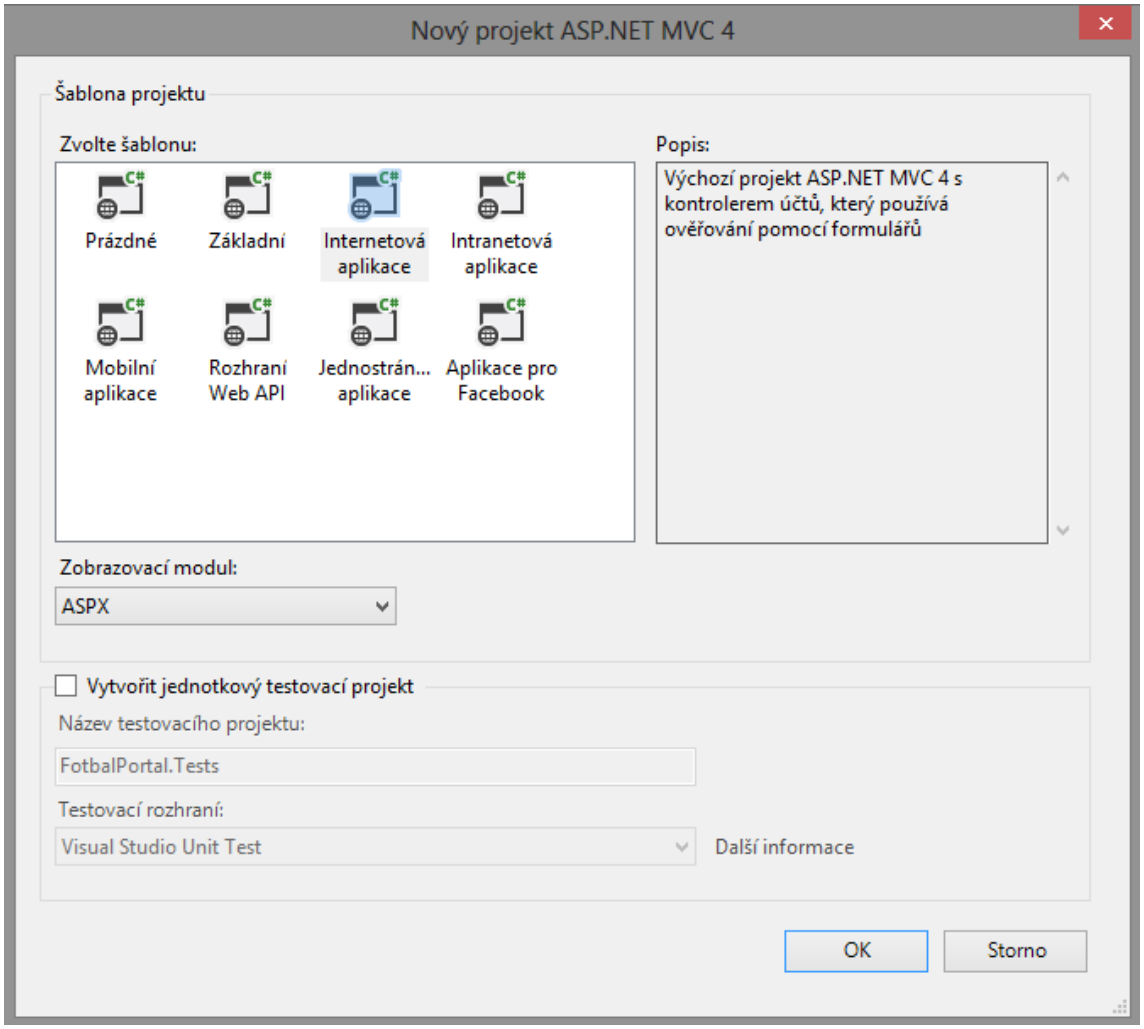
Prvním krokem v implementaci je založení nového projektu. Na začátku je nutné si rozmyslet jaké technologické náležitosti má naše aplikace splňovat. Ty již byly definovány v jiné kapitole. Visual Studio nabízí několik způsobů vytvoření nového projektu.



Obrázek 5.3: Vytvoření nového projektu

Na obrázku 5.3 je vidět, ze kterých typů webových aplikací se může vybírat při tvorbě nového projektu. V tomto případě bude použit .NET Framework 4.5 a MVC 4. Visual Studio ušetří práci a pomůže vytvořit základní kostru webu, která splňuje tyto specifikace. Stejně tak si vývojář může zvolit jinou verzi .NET Frameworku nebo MVC vzoru. Projekt lze vytvořit také úplně prázdný, kde si zkušený vývojář může specifikace nadefinovat sám.

VS nabízí také tvorbu tzv. serverových ovládacích prvků. Co tyto *Server Controls* jsou? Jsou to tagy, které jsou rozeznávané serverem. Rozeznávány jsou tři typy tagů – tradiční HTML tagy, nové ASP.NET tagy a tagy pro vstupní validaci u formulářů. Tyto prvky se používají hlavně u webových aplikací založených na ASP.NET Web Forms, ale lze je využít i pro MVC aplikaci. Prvek však nesmí využívat *ViewState*. [28]



Obrázek 5.4: Zvolení šablony

Po prvotní definici aplikace je na řadě další krok (obrázek 5.4). Zde se vybírá šablona projektu. Znovu je na výběr z několika typů, podle toho jaký je záměr s aplikací. Šablona „Internetová aplikace“ nabízí oproti základní šabloně kontroler pro uživatelské účty, takže není důvod si přidělávat práci a je možné ji zvolit. Ostatní typy neodpovídají definici webové aplikace, která bude vyvíjena.

V tomto případě není nutné vytvářet testovací projekt, ale v každém případě je nutné zvolit zobrazovací modul. VS v základu nabízí dvě možnosti – ASPX a Razor. Liší se ve způsobu zápisu zdrojového kódu do výstupních webových stránek (Views). ASPX je tradiční zobrazovací modul, drží se ASP syntaxe a koncovky souborů jsou *aspx*. Razor je oproti tomu novější zobrazovací modul, který byl vyvinut za účelem zpřehlednění zápisu kódu ve webových stránkách a jeho soubory mají koncovky *cshtml*.

Tyto zobrazovací moduly jsou založené na kódu, který je ve zdrojovém kódu view na straně serveru a generuje HTML kód. Protože je Razor technologie založená na ASP.NET, poskytuje její plný výkon a zvyšuje produktivitu zkušených vývojářů. Smysl Razoru je také v tom, že se snaží poskytnout lépe optimalizovaný HTML kód. V tomto projektu bude autor využívat z výukových důvodů oba zmíněné zobrazovací moduly. [29]

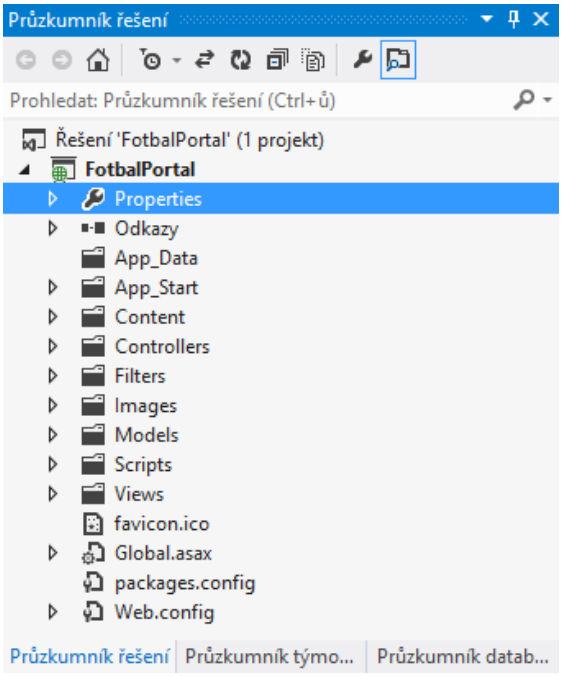
Příklad ASPX:

```
<ul id="items">

<% foreach(var i in items) { %>

<li><%=i.Name %></li>

<% } %>

</ul>
```

To samé v Razoru:

```
<ul id="items">

@foreach(var i in items) {

<li>@i.Name</li>

}

</ul>
```

Po dokončení druhého kroku je získána základní kostra projektu, ve které už je správně předdefinovaná adresářová struktura projektu v ASP.NET MVC 4 a vytvořené konfigurační soubory celé webové aplikace. Vývojové prostředí Visual Studia je velmi intuitivní a vývojář, který už programoval v jiném prostředí, nebude mít problém se rychle zorientovat. Nebudou zde zdlouhavě popisovány jednotlivé úkony ve VS, to nepatří do obsahu této práce. Nyní bude popsána již zmiňovaná adresářová struktura nové aplikace. Na následujícím obrázku 5.5 se podíváme, k čemu jednotlivé složky slouží.



Obrázek 5.5: Adresářová struktura nového projektu v ASP.NET MVC 4

Na obrázku výše je několik složek a souborů, ke kterým si uvedeme pár poznámek a vysvětlení:

- Properties – první složku interpretuje VS více jako konfigurační soubor. Jedná se o tu nejzákladnější konfiguraci celého projektu. Definuje se zde například použitá verze .NET Frameworku, typ výstupu aplikace, faviconu webu, nastavuje se zda se jedná o debug verzi aplikace, nastavuje se optimalizaci kódu, upozorňování na chyby nebo digitální podpisy a certifikáty.
- Odkazy – v této složce se nacházejí odkazy na všechny knihovny tříd, nejen .NET Frameworku, které aplikace používá.
- App_Data – slouží jako úložiště dat v případě, že aplikace nepoužívá databázový server. Data jsou pak uložena v jednotlivých databázových souborech.
- App_Start – zde je několik souborů, ve kterých jsou napsány třídy starající se o start aplikace. V těchto souborech se definuje například, jaké soubory s JavaScriptem se mají na stránce načítat nebo jaká domovská stránka má na webu být. Je zde také umístěn soubor, který umožňuje uživatelům přihlásit se na web pomocí jiné služby (Facebook, Twitter apod.).

- Content – složka pro obsahovou část webu. Nacházejí se zde kaskádové styly, grafické šablony webu a obrázky s nimi spojené.
- Controllers – první složka, která je specifická pro návrhový vzor MVC. Zde se nacházejí všechny kontrolery, které obstarávají logiku jednotlivých stránek webu.
- Filters – do této složky patří vlastní třídy, které přidávají specifické funkce před nebo po vykonání akce jednotlivých kontrolerů.
- Images – úložiště pro obrázky použité na webu.
- Models – další složka, která je bezprostředně spjata s MVC vzorem. Jak už název napovídá, jsou zde modely vázané na databázovou vrstvu.
- Scripts – adresář, který obsahuje různé javascriptové knihovny, zejména framework jQuery a jeho rozšíření.
- Views – poslední důležitá složka, alespoň co se z pohledu MVC týče. Zde jsou všechna zobrazení, která uvidí návštěvník na webové stránce. Jak jsme již popsali v kapitole o MVC vzoru, Views zobrazují data, která jim připraví Controller. Jejich strukturu definuje Model.
- Globals.asax – tento soubor startuje celou aplikaci, je to vlastně varianta souboru *index.html* u jednoduchých webových stránek. Nacházejí se zde především instance tříd z adresáře App_Start.
- Packages.config – jedná se o XML soubor, ve kterém jsou zaznamenány nainstalované knihovny tříd včetně jejich verzí.
- Web.config – další konfigurační XML soubor, který definuje jiné vlastnosti webu než soubor ve složce Properties. Jsou zde k nalezení deklarace Entity Frameworku, spojení na databázi nebo nastavení správy uživatelských účtů a rolí.

5.5 Databáze

V základním nastavení se data ukládají do jednotlivých databázových souborů. To není pro webovou aplikaci, která používá databázový server vhodné a je dobré si tyto záležitosti vyřešit hned na začátku vývoje. V této kapitole bude popsán Microsoft SQL Server 2012, který je aplikace na .NET Frameworku velice vhodný. Jedna podkapitola

se bude také týkat Entity Frameworku, který bude webová aplikace používat pro manipulaci s daty.

5.5.1 MS SQL Server 2012

Jak již bylo zmíněno, v aplikaci se použije SQL Server od společnosti Microsoft. Tento nástroj je také jeden z produktů, který Microsoft nabízí na ozkoušení studentům zcela zdarma, takže je naprosto ideální pro použití v ukázkové aplikaci.

SQL Server má dlouhou historii. Jeho první verze byla vydána v roce 1989 a fungovala pod operačním systémem OS/2. První verze MS SQL Serveru byly založené na kódu Sybase SQL Serveru, který zakoupil Microsoft od společnosti Sybase. Tím Microsoft vstoupil do světa relačních databází a vyhlásil konkurenční boj společnostem Oracle nebo IBM. Až verze 6.0 byla první, kterou Microsoft vyvinul sám, bez pomoci společnosti Sybase. Od té chvíle šly obě společnosti vlastní cestou a Sybase dokonce svůj produkt přejmenovala na Adaptive Server Enterprise, aby předešla jakékoliv záměně s produktem od Microsoftu.

Microsoft pracoval na dalších verzích, které byly lepší, použitelnější a umožňovaly širší využití. Verze 8.0 (MS SQL Server 2000) byla založená na architektuře klient/server, škálovatelná, bezpečná a měla integrovanou podporu jazyka XML. Tato verze byla také navržena pro zvládání velkého objemu transakcí, podporovala replikaci, díky které bylo možné udržovat více kopií dat včetně kontroly synchronizace.

Nástupcem MS SQL Serveru 2000 se v říjnu roku 2005 stal MS SQL Server 2005. SQL Server 2005 zpracovává online transakce (OLTP) ve velkém množství, umožňuje uchovávat informace v datových skladech a nabízí také širokou škálu možností pro analýzu, integraci a reportování obchodních dat. S touto verzí přišlo také SQL Management Studio, které výrazně zefektivňuje a ulehčuje práci s databázemi tím, že umožňuje spravovat databázový stroj, služby pro analýzu, reportingové servery a řídit integrační procesy.

O tři roky později se na trhu objevila nová verze databázového systému MS SQL Server 2008. Svého předchůdce vylepšuje a rozšiřuje o nové nástroje a funkce, které

dokážou například zachytávat události ze serveru nebo konfigurovat databázový server bez expertních znalostí.

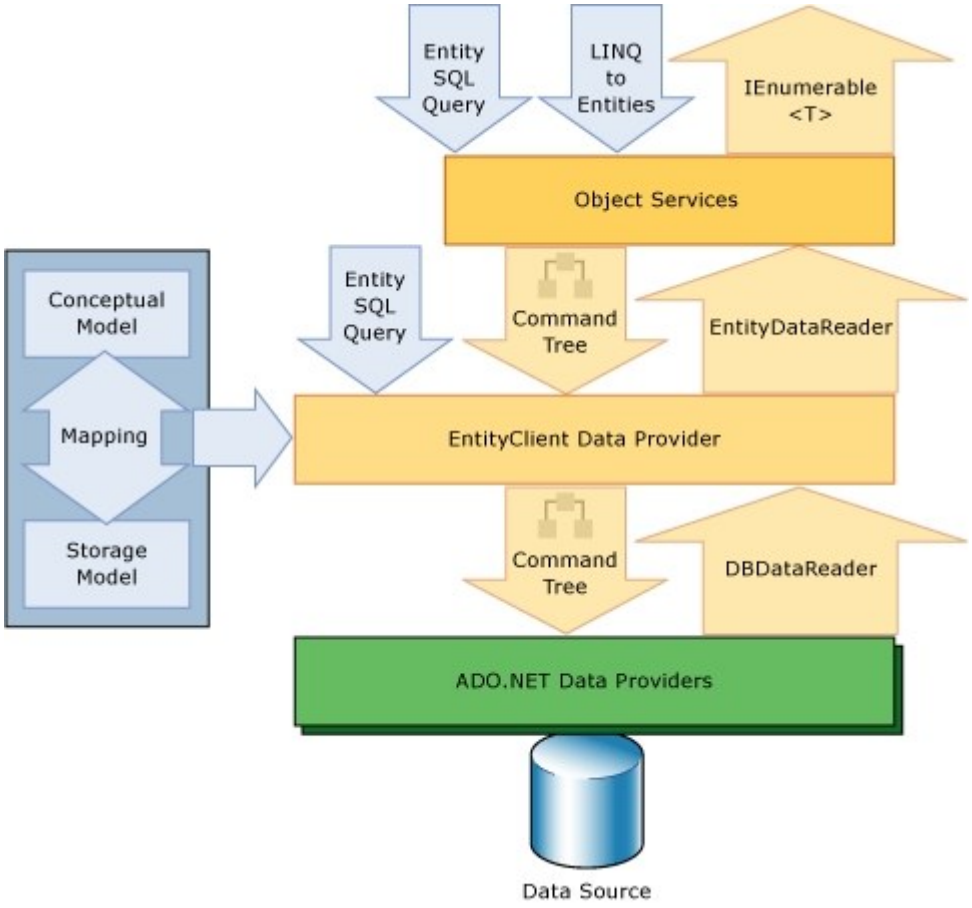
Pravidelné uvedení nové verze SQL Studia pokračovalo na konci roku 2011, kdy byl představen MS SQL Server 2012. Nová verze s sebou přinesla znovu nové funkce a nástroje, které vylepšují dostupnost a konfiguraci databází. [32]

5.5.2 Entity Framework

Pokud webový vývojář nepoužívá žádné objektově relační mapování, je nucen sestavovat SQL dotazy, které mu teprve vrací ty správná data. Tato technika rozhodně není vhodná pro tvorbu rozsáhlé aplikace. Tato práce se zabývá tvorbou webové aplikace v .NET, a alespoň na této platformě je použití frameworku pro práci s databází skoro až nutností. Při tvorbě nového projektu nám dokonce Visual Studio připraví náš projekt tak, aby používal Entity Framework.

Entity Framework (celým názvem ADO.NET Framework, dále jen EF) je jeden ze spousty podobných frameworků, které se liší použitím na různé programovací jazyky a platformy. Pro jazyk *Java* se jedná například o *Hibernate* nebo *Java Persistence API*. V případě platformy .NET existuje také několik ORM frameworků (*NHibernatem*, *MyBatis* a další), ale EF je již v platformě integrováno. Cílem těchto frameworků je snížit množství kódu a zajistit tak lepší údržbu aplikace. Použití EF nabízí několik výhod:

- Aplikace je zcela oproštěna od závislosti na zdrojovém kódu, který pracuje s SQL dotazy nebo logikou starající se o schéma databáze.
- Mapování lze změnit bez velkého zásahu do zdrojového kódu.
- Vývojáři pracují s konzistentním modelem databáze, která může být aplikována na různé databázové stroje a prostředí.
- Lze použít několik konceptuálních modelů, které jsou nezávislé na databázovém prostředí a mohou být mapovány na jedno databázové schéma. [31]



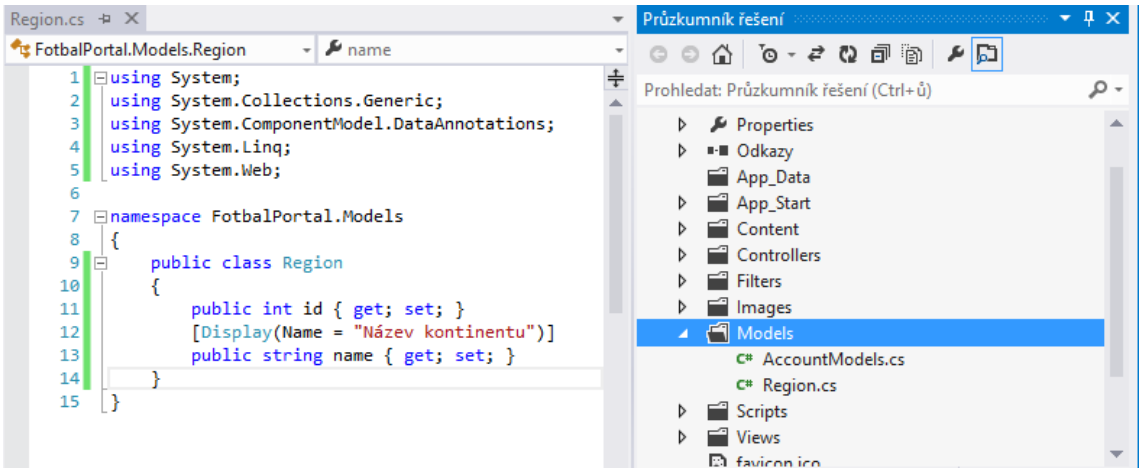
Obrázek 5.6: Architektura EF pro přístup k datům [38]

Na obrázku 5.6 je ilustrována architektura EF pro přístup k datům. Je zde názorně předvedeno, jak probíhá celý dotazu a následného získání dat z databáze. Na začátku celého procesu je dotaz, který může přijít v podobě Entity SQL (případně LINQ). Vývojář zde nepíše žádný SQL kód, vše se definuje pomocí objektů. Tyto objekty přichází na řadu hned vzápětí v podobě Object Services. Tato komponenta programové vrstvy zpracuje dotaz a předává ho dále EntityClient Data Provideru. EntityClient Data Provider je již skutečná součást Entity Frameworku v tomto procesu. V tomto místě mapovací framework zpracovává data, tak aby je uměl přiřadit ke konkrétním tabulkám. Na konkrétní tabulky mapuje konceptuální model. Konceptuální model je definovaný v jednotlivých třídách, které budou popsány v dalších částech této práce. Výsledek se posílá ADO.NET vrstvě, která přesně ví, ke kterým tabulkám přistupovat a jaká data má vrátit. Data vrací komponenta DBDataReader, která je předá zpátky na EntityClient Data Provider. Zde mapovací framework přiřadí databázová data ke konceptuálnímu modelu, tak aby s nimi aplikace mohla pracovat a zobrazit je. Object Services si data

vytáhne komponentou EntityDataReader a aplikaci je poskytne v kolekci přes rozhraní IEnumerable.

5.6 Model, View, Controller

V této části bude v praxi ukázáno jak pracovat s MVC návrhovým vzorem. Jak bylo popsáno v jedné z předchozích kapitol, všechny části MVC mají své jasné místo v adresářové struktuře. Tvorba webové aplikace ve Visual Studiu v souladu s návrhovým vzorem MVC má svou jasnou posloupnost. Nejprve se vytvoří třída, která představuje *Model*. Bude uveden příklad, kdy se vytvoří dvě třídy, které mají mezi sebou vztah. Jako první *Model* vlastní výroby se zvolí třídy pro označení zeměpisného původu objektů v databázi. Půjde o vztah mezi kontinenty a zeměmi. Tato návaznost se bude později používat při tvorbě a úpravě klubů, soutěží a hráčů.

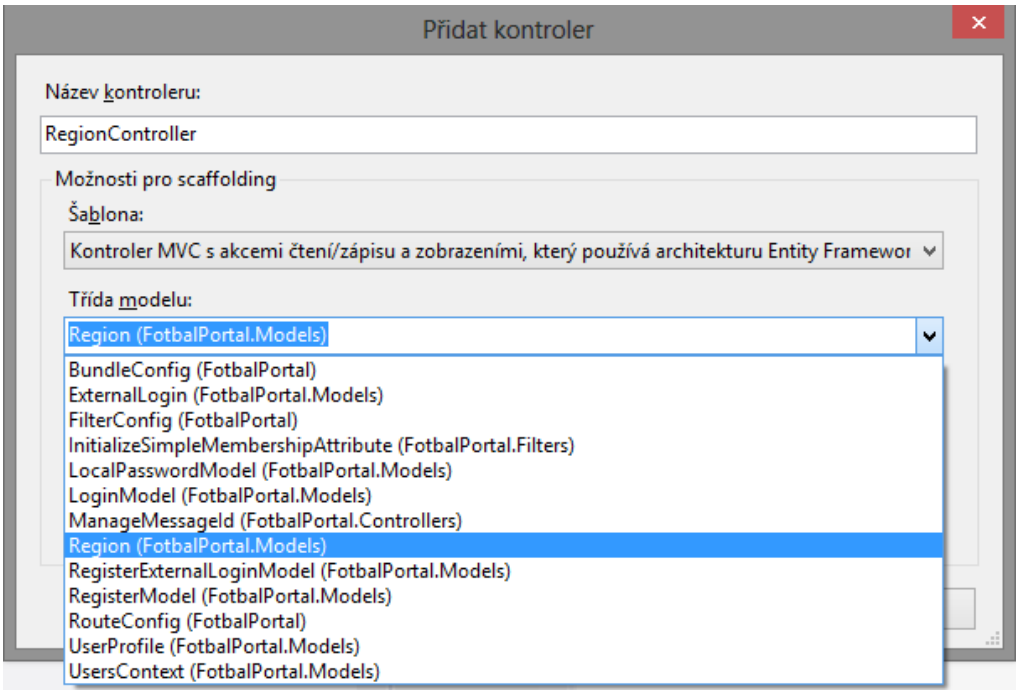


Obrázek 5.7: Nová třída v projektu

Na obrázku 5.7 je vidět nová třídu, která nese název *Region*. V pravé části obrázku je znázorněna adresářová struktura, kde je vidět, že nová třída se opravdu zařadila do složky *Models*, ve které se budou uchovávat všechny třídy, které mají být vázané na některou z tabulek v databázi. V levé části je zdrojový kód třídy v programovacím jazyce C#. Je to vlastně jen jednoduchá definice třídy, ve které je řečeno, jaký má třída název a jaké parametry bude obsahovat budoucí tabulka. V tomto případě se jedná pouze o identifikační číslo (id) a název kontinentu. Pro název kontinentu jsou rovnou použití i tzv. anotace. Anotace mají spoustu funkcí a vlastnosti,

ale v této chvíli bude stačit skutečnost, že pomocí vlastnosti *Display* se určí, pod jakým názvem se bude vlastnost *name* zobrazovat ve formulářích.

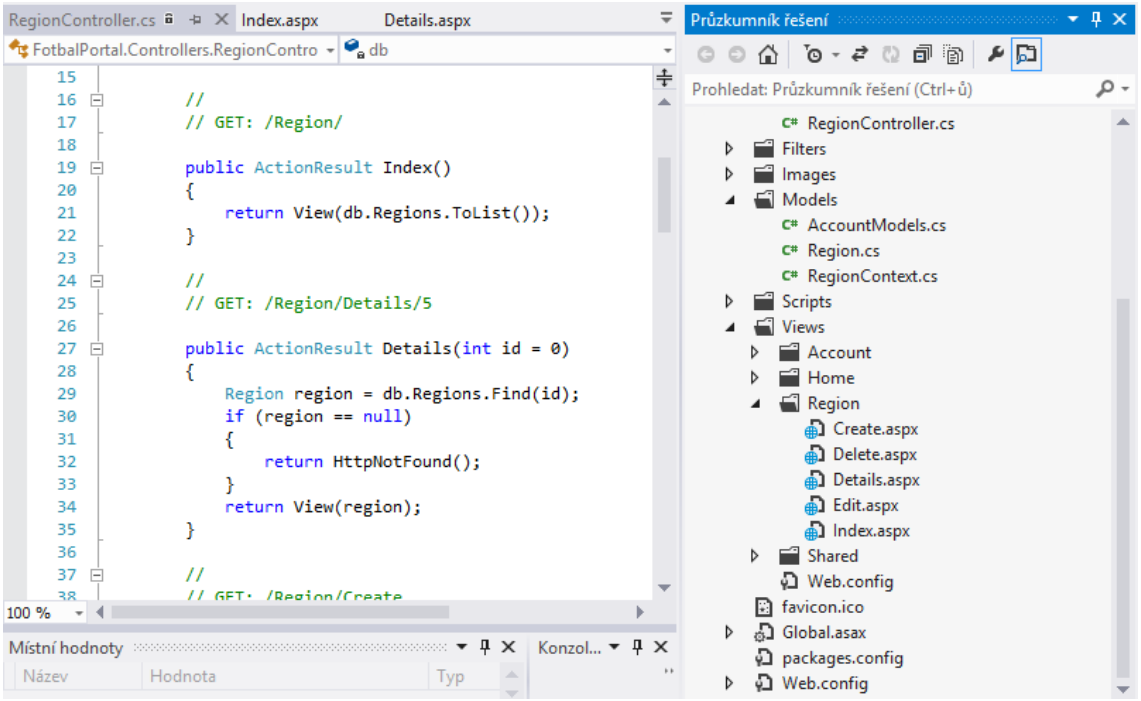
V této chvíli je první model hotový. Jako další krok je vytvoření kontroleru (*Controller*). Kontroler, který je také vlastně další třída, bude vytvořen v adresáři *Controllers*, ale ještě předtím než se začne cokoliv dalšího dělat, je nutné projekt zkompilovat. Tím se zajistí, že se vytvořený model přidá do knihoven projektu. V momentě, kdy je zakládán nový kontroler, Visual Studio do nabídky tříd, ze kterých kontroler vychází, nabídne i nově vytvořený model.



Obrázek 5.8: Tvorba kontroleru, který využívá architekturu Entity Framework

Na obrázku 5.8 je vidět, že Visual Studio šetří práci tím, že nabídne vytvoření kontroleru přímo s použitím architektury Entity Frameworku. Kromě tohoto poměrně přátelského přístupu je možné vždy vytvořit prázdný kontroler a všechny základní vazby, které jinak Visual Studio vytvoří, definovat ručně.

Zároveň není nutné se starat o vytvoření základních pohledů (*Views*) pro tento kontroler. Visual Studio automaticky vytvoří složku pro každý nový model. Tu lze nalézt v adresáři pro pohledy a je už jen na vývojáři jak bude s připravenými pohledy pracovat.



Obrázek 5.9: adresářová struktura po vytvoření nového kontroleru

Automaticky se vytvoří celkem pět pohledů, jejichž funkci výstižně popisují názvy:

1. Create.aspx – formulář pro vytvoření
2. Delete.aspx – stránka pro smazání
3. Details.aspx – detail entity
4. Edit.aspx – formulář pro úpravu
5. Index.aspx – stránka, na které nalezneme základní výpis entit

V levé části obrázku 5.9 je vidět část zdrojového kódu nového kontroleru. Jde o metody typu *ActionResult*, které se automaticky mapují na URL adresy webové aplikace. První metoda Index() se vyvolá, pokud uživatel webu vstoupí na URL adresu /region/. Druhá metoda Details() s parametrem identifikátoru vrátí detail konkrétní entity. Ta pod URL /region/details/5 vybere z databáze entitu, která má identifikační číslo 5 a zobrazí jeho detail s podrobnostmi.

Stejným způsobem, jakým byly zaneseny do systému třídu pro kontinenty, se zanese i třída pro země s názvem *Country*. Kromě názvu bude jediný rozdíl v návrhu

třídy, protože každá země patří na některý kontinent. Bude tam tedy vazba na jinou třídu a implementace toho je ukázána na následujícím obrázku 5.10.

```
namespace FotbalPortal.Models
{
    public class Country
    {
        public int id { get; set; }
        [Display(Name = "Název země")]
        public string name { get; set; }
        [Display(Name = "Kontinent")]
        public int regionID { get; set; }
        public virtual Region region { get; set; }
    }
}
```

Obrázek 5.10: Třída modelu Country

Ve třídě *Country* je kromě parametrů pro identifikační číslo a název země i parametr *regionID*. Sem se bude ukládat ID kontinentu, na který země patří. Zároveň je tam ještě virtuální proměnná *region* stejnojmenného typu. Nebude sice součástí databázové struktury, ale při každém načtení instance země se do této proměnné automaticky uloží instance příslušného kontinentu pro případ, že by bylo potřeba s ní pracovat.

5.7 Migrations

V průběhu vývoje webové aplikace se s vysokou pravděpodobností několikrát stane, že bude nutné upravovat databázovou strukturu. Ať už jde o přidání nových tabulek nebo o úpravu stávajících. V této situaci není žádoucí manipulovat s databází přímo a dodatečně provádět SQL dotazy, ale je nutné použít mnohem sofistikovanější techniku.

Entity Framework pro tyto účely nabízí tzv. migrace (*Migrations*). Jedná se o updaty na databázi, které se sestavují automaticky podle existujících modelů. Migrace se evidují v projektu a není problém je dohledat v historii a některou migraci vrátit zpátky. S migracemi se pracuje pomocí příkazů v nástroji *Package Manager Console*, který je součástí Visual Studia.

Aby projekt mohl využívat migrace, je nejprve nutné je povolit. To se provede příkazem *Enable-Migrations -ContextTypeName [ContextClassName]*, kde se místo proměnné *ContextClassName* napíše název třídy, která komunikuje s databázovou

vrstvou a obsahuje databázové sety pro každý model v projektu (v tomto případě se nazývá *PortalContext*). Po úspěšném proběhnutí tohoto příkazu je projekt připravený k využívání výhod migrací.

Migrace jsou úzce spjaté s MVC architekturou projektu. Pro úspěšné proběhnutí je tedy nutné vytvořit nový model včetně kontroleru a zahrnout jeho použití do databázového kontextu (tzn. třída, nad kterou se v předchozím odstavci zapnula migrace). Následně se musí projekt zkompilovat a až poté je možné do příkazového řádku *Package Manager Console* vepsat příkaz *Add-Migration [Name]* s názvem migrace. V předchozí kapitole bylo popsáno vytvoření jednoduchého modelu, kontroleru a view pro kontinenty. Na následujícím obrázku je ilustrováno, jak vypadá migrace vytvořená pro přidání tabulky pro tento model do databáze.

```
namespace FotbalPortal.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class addRegion : DbMigration
    {
        public override void Up()
        {
            CreateTable(
                "dbo.Regions",
                c => new
                    {
                        id = c.Int(nullable: false, identity: true),
                        name = c.String(),
                    })
                .PrimaryKey(t => t.id);
        }

        public override void Down()
        {
            DropTable("dbo.Regions");
        }
    }
}
```

Obrázek 5.11: Migrace pro přidání tabulky kontinentů do databáze

Na obrázku 5.11 je vidět, že migrace obsahuje kód pro zpracování i vrácení migrace. Samotná realizace zpracování se provede pouze příkazem *Update-Database*

v konzoli. Na konci tohoto procesu je v databázi nová tabulka pro evidenci kontinentů a vývojář tím využívá výhody Entity Frameworku.

5.8 Autentizace a autorizace

Aplikační frameworky se používají pro vývoj aplikací kvůli výhodám a ulehčením, které z jejich použití plynou. Jednou z výhod ASP.NET frameworku je také to, že proces autorizace a autentizace je zde částečně vyřešený a zahrnutý již v základním projektu, který vygeneruje Visual Studio. V rámci implementace jsou k dispozici mechanismy pro přihlášení a ověření uživatele. K tomu framework používá knihovnu *Membership*. Zároveň s vestavěnou podporou rolí jde o téměř hotové řešení.

Pro úplný administrátorský komfort chybí správa rolí a jejich přidělení uživatelům, jejíž implementace bude ukázána. Pro správu uživatelských rolí je nutné vytvořit novou akci v kontroleru pro uživatelské účty (*AccountController*). Akce bude mít název *RoleIndex* a bude se jednat o výpis všech rolí v systému.

```
[Authorize(Roles = "Admin")]

public ActionResult RoleIndex()

{
      var roles = Roles.GetAllRoles();
      return View(roles);
}
```

Zdrojový kód 5.1: Ukázka předání výpisu rolí do view

Nové view pro tuto akci bude mít následující podobu:

```
@{
    ViewBag.Title = "Uživatelské role";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

@Html.ActionLink("Vytvořit novou roli", "RoleCreate") |
@Html.ActionLink("Přidat roli uživateli", "RoleAddToUser")
<h2>Uživatelské role</h2>
<table>
    @foreach (string s in Model)
    {
        <tr>
            <td>@s</td>
            <td>
                <span onclick="return confirm('Opravdu smazat?')">
                    <a href="/Account/RoleDelete?RoleName=@s"
class="delLink">Smazat</a>
                </span>
```

```
            </td>

        </tr>
    }

</table>
```

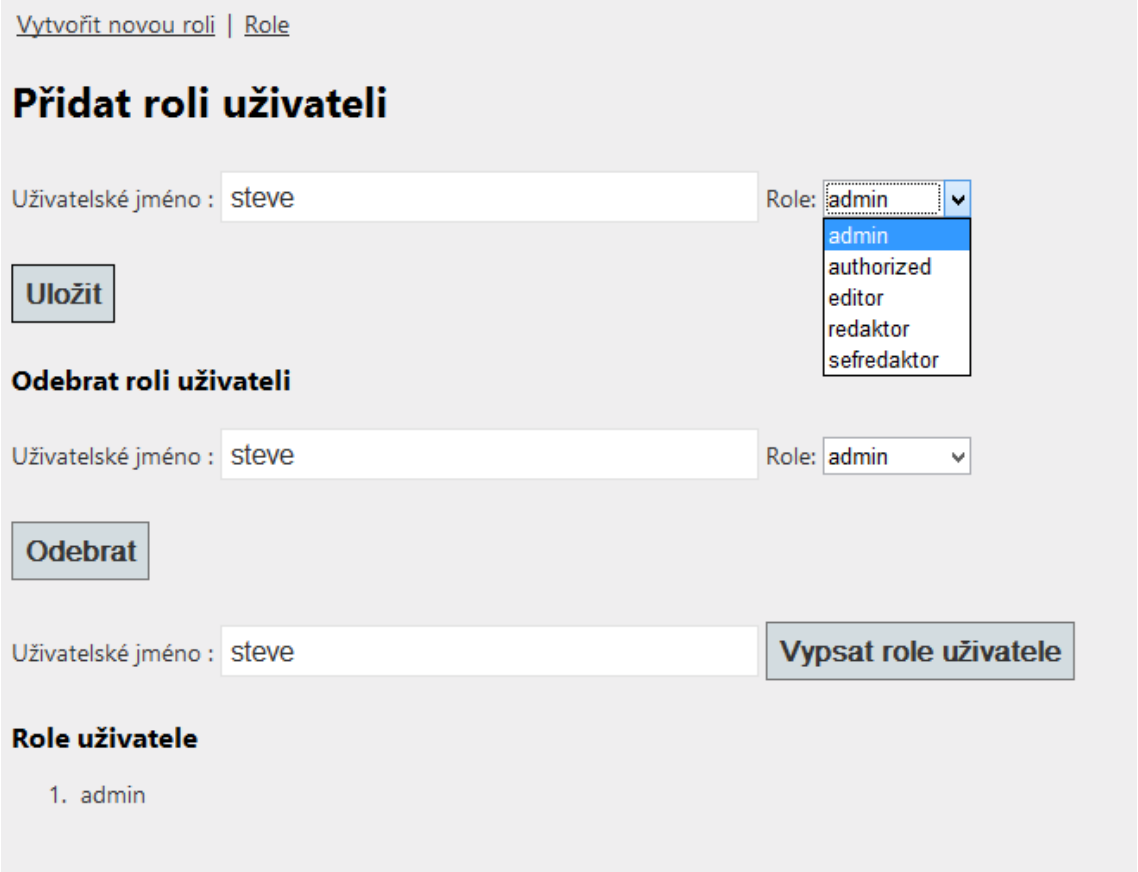
Zdrojový kód 5.2: View pro seznam uživatelských rolí

V tomto view, které je reprezentováno zdrojovým kódem 5.2, jsou k dispozici i odkazy pro vytvoření nové role a přidání role k uživateli, jejichž akce budou rozebrány dále v textu. Každou roli lze také smazat. V odkazu pro smazání se předává parametr v podobě názvu uživatelské role. Tato akce je bez view a zpracuje se pouze vykonáním kódu v kontroleru, které je vidět ve zdrojovém kódu 5.3:

```
[Authorize(Roles = "Admin")]
public ActionResult RoleDelete(string RoleName)
{

      Roles.DeleteRole(RoleName);
      return RedirectToAction("RoleIndex", "Account");

}
```

Zdrojový kód 5.3: Vykonání smazání uživatelské role

Pro správu rolí konkrétního uživatele je potřeba vytvořit komplexnější view, kde je možné roli přidat, odebrat a také vypsat seznam rolí pro uživatele, se kterým se právě pracuje. Výsledek bude vypadat jako níže na obrázku 5.12.



Obrázek 5.12: Komplexní view pro správu rolí uživatele

View na předchozím obrázku je vytvořené primárně pro přidání role konkrétnímu uživateli, který se vyhledá dle uživatelského jména. Do toho samého view jsou ale také přidány dva další formuláře, jeden pro odebrání role, druhý pouze pro výpis rolí. Veškerý výkonný kód se provádí v kontroleru pro uživatelské účty. Přidání role uživateli je demonstrováno v následujícím zdrojovém kódu 5.4.

```
[Authorize(Roles = "Admin")]
public ActionResult RoleAddToUser()
{
      SelectList list = new SelectList(Roles.GetAllRoles());
      ViewBag.Roles = list;

      return View();
}

[Authorize(Roles = "Admin")]
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult RoleAddToUser(string RoleName, string UserName)
{
if (Roles.IsUserInRole(UserName, RoleName))
{
      ViewBag.ResultMessage = "Uživatel již v této roli je!";
```

```
}
else
{
      Roles.AddUserToRole(UserName, RoleName);
      ViewBag.ResultMessage = "Role přidána.";
}
      SelectList list = new SelectList(Roles.GetAllRoles());
      ViewBag.Roles = list;
      return View();
}
```

Zdrojový kód 5.4: Ukázka vykonání akce pro přidání role uživateli v kontroleru

První akce *RoleAddToUser* je počáteční načtení view pro přidání role. Kontroler předává do view také seznam rolí v kolekci typu *SelectList*. Ten se předává v proměnné *ViewBag*, která v MVC architektuře funguje jako balíček, který předává jakékoliv proměnné z kontroleru do view.

Druhá akce *RoleAddToUser*, která se vykoná při odeslání formuláře, nejprve dle předaných parametrů zkontroluje, zda uživatel již má danou roli. Pokud ano, skript nevykoná přidání a pouze nastaví do proměnné *ViewBag* hlášku uživateli. Pokud uživatel předávanou roli nemá, přidá se do seznamu jeho rolí a kód znovu vrátí view.

```
<h2>Přidat roli uživateli</h2>

@using (Html.BeginForm("RoleAddToUser", "Account"))
{
    @Html.AntiForgeryToken()
    @Html.ValidationSummary(true)

    <div class="message-success">@ViewBag.ResultMessage</div>
    <p>
        Uživatelské jméno : @Html.TextBox("UserName")
        Role: @Html.DropDownList("RoleName", ViewBag.Roles as SelectList)

    </p>

    <input type="submit" value="Uložit" />

}
```

Zdrojový kód 5.5: View pro přidání role uživateli

Ve zdrojovém kódu 5.5 je vidět implementace formuláře ve View. Pro formulář, který se stará o odebrání uživatele z určité role se použije totožný kód s tím rozdílem, že v řádku, který obsahuje metodu *Html.BeginForm* se jako první parametr místo *RoleAddToUser* předá *DeleteRoleForUser* jehož kód v kontroleru vypadá následovně:

```
[HttpPost]
[Authorize(Roles = "Admin")]
```

```
[ValidateAntiForgeryToken]
public ActionResult DeleteRoleForUser(string UserName, string RoleName)
{
      if (Roles.IsUserInRole(UserName, RoleName))
      {
            Roles.RemoveUserFromRole(UserName, RoleName);
            ViewBag.ResultMessage = "Role odebrána!";
      }
      else
      {
            ViewBag.ResultMessage = "Tuto roli uživatel nemá.";
      }

      ViewBag.RolesForThisUser = Roles.GetRolesForUser(UserName);
      list = new SelectList(Roles.GetAllRoles());
      ViewBag.Roles = list;
      return View("RoleAddToUser");
}
```

Zdrojový kód 5.6: Odebrání role uživateli v kontroleru

Zdrojový kód 5.6 se spouští při odeslání formuláře pro odebrání role. Zde není potřeba žádná další akce, která definuje view, protože to už dělá akce *RoleAddToUser*, na kterou se po proběhnutí přesměruje tento kód.

Poslední částí toho view je formulář s výpisem rolí k určitému uživateli. Na straně kontroleru je následující kód:

```
[Authorize(Roles = "Admin")]
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult GetRoles(string UserName)
{
      if (!string.IsNullOrWhiteSpace(UserName))
      {
            ViewBag.RolesForThisUser = Roles.GetRolesForUser(UserName);
            SelectList list = new SelectList(Roles.GetAllRoles());
            ViewBag.Roles = list;
      }
      return View("RoleAddToUser");
}
```

Zdrojový kód 5.7: Předání všech rolí vybraného uživatele do view

Ve zdrojovém kódu 5.7 je ukázáno vytáhnutí seznamu rolí dle uživatelského jména a jeho předání v proměnné *ViewBag* do view. Formulář s výpisem bude mít následující podobu:

```
@using (Html.BeginForm("GetRoles", "Account"))
{
    @Html.AntiForgeryToken()
    <p>
        Uživatelské jméno : @Html.TextBox("UserName")
        <input type="submit" value="Vypsat role uživatele" />
```

```
    </p>
}

@if (ViewBag.RolesForThisUser != null)
{
    <h3>Role uživatele</h3>
    <ol>
        @foreach (string s in ViewBag.RolesForThisUser)
        {
            <li>@s</li>
        }
    </ol>

}
```

Zdrojový kód 5.8: Formulář pro výpis všech rolí vybraného uživatele

V kontroleru je u každé akce nad názvem řádek *[Authorize(Roles="Admin")]*. Jedná se o vestavěnou autorizaci pro MVC model. Pokud se tento řádek nespecifikuje, do view se dostane kterýkoliv návštěvník webu. V momentě, kdy se přidá specifikace rolí, systém do view pustí jen uživatele, kteří je mají přidělené.

5.9 Menu

V tomto projektu jsou dva druhy menu. První menu se zakládá na kategoriích, které odpovídají národním soutěžím v zemích, kde hrají kluby. V rámci těchto kategorií se vydávají články a návštěvník si může prohlížet sestavy a výsledky klubů a statistiky jejich hráčů. V této a dalších kapitolách nebude znovu popisováno vytvoření a zavedení nového modelu, kontroleru a view. Vše potřebné je již popsané v podkapitolách 5.6 a 5.7. Více se budou rozebírat pouze rozdíly, kdy se implementace liší od standardního postupu nebo od výchozích souborů, které jsou vygenerované Visual Studiem.

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Web;

namespace FotbalPortal.Models
{
    public class Category
    {
        public int id { get; set; }
        [Display(Name = "Název kategorie")]
        public string name { get; set; }

        [Display(Name = "Nadřazená kategorie")]
        public int? parentID { get; set; }
        public virtual Category parent { get; set; }
    }
}
```

Obrázek 5.13: Model pro kategorie

Na obrázku 5.13 je struktura modelu pro kategorie. Jedná se o jednoduchý model, který obsahuje pouze identifikátor, název kategorie a odkaz na případnou nadřazenou kategorii. Pro parametry *name* a *parentID* jsou zde také použité anotace, které v tomto případě stanovují popis datového pole, které se použije v každém view. Datový typ u *parentID* je definován včetně otazníku, který znamená, že tato hodnota je nepovinná. Na posledním řádku je virtuální hodnota pro nadřazenou kategorii, jejíž funkci a význam byl popsán již dříve.

Druhý typ menu se plní ručně a je založený na klasické struktuře Model-View-Controller. Jedná se o všechny další odkazy, které nesouvisí s národními soutěžemi. Třída pro tyto položky se nazývá *MenuItem* a má atributy *id*, *title* (popisek) a *path* (cesta v systému). Menu je následně vloženo jako view do šablony celého projektu.

5.10 Články a komentáře

Nedílnou součástí informačního systému jsou články a možnost komentářů. Články vkládají do systému redaktoři a podléhají schválení od šéfredaktorů. Běžní návštěvníci je mohou komentovat a vyjádřit svůj názor, který ale může být v případě nevhodného obsahu cenzurován.

```
namespace FotbalPortal.Models
{
    public class Article
    {

        public int id { get; set; }
        [Display(Name = "Autor")]
        public int author { get; set; }
        [Display(Name = "Stav")]
        public int status { get; set; }
        [Display(Name = "Titulek")]
        public string title { get; set; }
        [Display(Name = "Text")]
        public string text { get; set; }
        [Display(Name = "Datum vytvoření")]
        public DateTime created { get; set; }
        [Display(Name = "Datum úpravy")]
        public DateTime? edited { get; set; }
        [Display(Name = "Datum zveřejnění")]
        public DateTime? published { get; set; }
        [Display(Name = "Komentáře povoleny")]
        public bool commentsAllowed { get; set; }

        [Display(Name = "Kategorie")]
        public int categoryID { get; set; }
        public virtual Category category { get; set; }

        [Display(Name = "Obrázek")]
        public string image { get; set; }

    }
}
```

Obrázek 5.14: Model článku

Na obrázku 5.14 je třída pro model článku. Článek nese informace o názvu, autorovi, stavu (zda je schválený), datu vytvoření, editace a zveřejnění, informaci o povolení komentářů, kategorii a textu. Ke článku je také možnost nahrát průvodní obrázek. Kategorie článku odpovídá kategoriím, které fungují také jako první z typů menu, které bylo popsané v předchozí kapitole. Články se tedy dělí podle toho, jaké národní soutěže se týkají a následně se objevují ve výpisu článků pod jednotlivými kategoriemi. O výpis článků v kategorii se stará akce *Category* v kontroleru pro články, která přebírá z URL adresy ID kategorie:

```
private PortalContext db = new PortalContext();

public ActionResult Category(int id = 0)
{
      ViewBag.categoryID = id;
```

```
        var articles = db.Articles.Include(a => a.category)
              .Where(a => a.categoryID == id)
              .Where(a => a.status == 1)
              .Where(a => a.published < DateTime.Now)
              .OrderByDescending(a => a.published);
       return View(articles.ToList());
}
```

Zdrojový kód 5.9: Předání článků zvolené kategorie do view

Zdrojový kód 5.9 ukazuje, jak kontroler vybere z databáze všechny články, které spadají do kategorie, jsou zveřejněné (status nastaven na hodnotu 1) a jejich datum a čas určený pro publikaci je starší než aktuální moment. Identifikační číslo kategorie se předá do view proměnnou *ViewBag*. Tam bude použito pro vytvoření odkazu na tabulku soutěže. Tabulky a statistiky popíšeme v jedné z následujících podkapitol.

```
<%@ Page Title="" Language="C#"
MasterPageFile="~/Views/Shared/Site.master"
Inherits="System.Web.Mvc.ViewPage<IEnumerable<FotbalPortal.Models.Article>
>" %>

<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent"
runat="server">
    Sportovní portál
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="MainContent"
runat="server">

<%: Html.ActionLink("Tabulka soutěže", "Table", "Result", new
{categoryID=ViewBag.categoryID }, null)%>

<% foreach (var item in Model) { %>
    <div class="article">
        <div class="image"><img src="<%: Url.Action("ImageResize", new {
width = 150, height = 150, filename = item.image } )%>" /></div>
        <div class="title"><a href="<%: Url.Action("Read", new {
id=item.id })%>"><%: Html.DisplayFor(modelItem => item.title) %></a></div>
        <% var length = item.text.Length;
           if (length > 200) length = 200; %>
        <div class="text"><%: Html.Raw(item.text.Substring(0,length) +
"..." ) %></div>
        <%: Html.DisplayFor(modelItem => item.published) %>
        <%: Html.DisplayFor(modelItem => item.category.name) %>
    </div>
<% } %>

</asp:Content>
```

Zdrojový kód 5.10: View pro výpis článků

Ve zdrojovém kódu 5.10 je ukázáno, jak probíhá výpis článků. Nad obsahem se pomocí metody *Html.ActionLink* vloží odkaz na tabulku soutěže, o kterou se stará kontroler výsledků, který zatím nebyl popsán. Ve foreach cyklu je pak zpracování článku.

Nejprve se zpracuje obrázek článku, který může být vložen v různé velikosti. Cílem je velikost upravit na rozměr 150x150 pixelů. Pro tento účel je v kontroleru metoda *ImageResize*, které se předává obrázek článku, a ta vytvoří jeho upravenou verzi:

```
public void ImageResize(int width, int height, string filename)
{
      var imageFile = Path.Combine(Server.MapPath("~/Images"), filename);
      WebImage image = new WebImage(imageFile).Resize(width,
height).Write();

}
```

Zdrojový kód 5.11: Metoda pro změnu velikosti obrázku

Metoda je bez návratové hodnoty (*void*) používá třídu *WebImage* pro vytvoření a přímé vložení obrázku v daném rozměru. Další zpracování článku je již jednoduché vypsání vybraných informací (název, datum zveřejnění, kategorie) a zkrácení textu článku na 200 znaků metodou *Substring*.

V detailu článku je možné komentovat. Aby bylo možné vypsat komentáře ve view někde pod článkem, je nutné je do view předat. V tomto případě se zkusí místo proměnné *ViewBag* použít proměnnou *ViewData*:

```
public ActionResult Read(int id = 0)

{
      Article article = db.Articles.Find(id);
      if (article == null)
      {
            return HttpNotFound();
      }

      var comments = db.Comments.Where(a => a.articleID == id).Where(a =>
a.disabled  == false);
ViewData["comments"] = comments.ToList();

return View(article);
}
```

Zdrojový kód 5.12: Předání instance článku ke čtení pro view Read

ViewBag a *ViewData* jsou oba objekty, přes které lze předat data z kontroleru do view. Technicky jsou všechna data přenášena přes *ViewData*, ale pokud použijeme *ViewBag* uděláme tím kolem *ViewData* dynamický obal, který z toho udělá objekt s atributy. Použitím *ViewData* jako ve zdrojovém kódu výše předáváme indexované pole dat.

```
<%@ Page Title="" Language="C#"
MasterPageFile="~/Views/Shared/Site.Master"
Inherits="System.Web.Mvc.ViewPage<FotbalPortal.Models.Article>" %>
```

```
<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent"
runat="server">
    <%: Html.DisplayFor(model => model.title) %>
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="MainContent"
runat="server">

<h2><%: Html.DisplayFor(model => model.title) %></h2>
<br />
<div class="read-article">
    <div class="image"><img src="/images/<%: Html.DisplayFor(model =>
model.image) %>" /></div>
    <div class="text">
        <%: Html.DisplayFor(model => model.text) %>
    </div>
</div>
<br />
<div class="clear-both"></div>

<div class="fb-share-button" data-href="
http://ehlousek.aspone.cz/Article/Read/1" data-
layout="button_count"></div>

<a href="https://twitter.com/share" class="twitter-share-button" data-
via="Steve5CZ">Tweet</a>

<script>!function(d,s,id){var
js,fjs=d.getElementsByTagName(s)[0],p=/^http:/.test(d.location)?'http':'ht
tps';if(!d.getElementById(id)){js=d.createElement(s);js.id=id;js.src=p+':/
/platform.twitter.com/widgets.js';fjs.parentNode.insertBefore(js,fjs);}}(d
ocument, 'script', 'twitter-wjs');</script>

<div class="g-plus" data-action="share"></div>

<div class="comments">
    <%
        if (User.Identity.IsAuthenticated)
        { %>
          <%: Html.Action("Create", "Comment", new { articleID =
Model.id})  %>
       <% }
    %>
    <%: Html.Partial("Comments", ViewData["comments"]) %>
</div>

</asp:Content>
```

Zdrojový kód 5.13: View pro detail článku

Ve zdrojovém kódu 5.13 je view pro detail článku. Pod výpisem nadpisu, fotografie a textu jsou tlačítka pro sdílení na sociálních sítích (Facebook, Twitter, Google Plus). Jedná se o vygenerovaná tlačítka, která poskytují sociální sítě webovým administrátorům, a každý článek je tedy možné sdílet. Ve spodní části kódu je část pro komentáře. Nejprve je nutné ověřit uživatele, zda je přihlášení proměnnou

User.Identity.IsAuthenticated, protože komentovat mohou jen registrovaní uživatelé. V tomto případě se role uživatele neověřuje anotací v kontroleru, ale až při výpisu view. Formulář pro vložení komentáře je totiž vestavěný ve view pro zobrazení článku a kdyby se ověřovala role v kontroleru, zákaz přístupu by platil pro celý článek i pro nepřihlášené uživatele. Pokud je uživatel registrovaný a přihlášený metoda *Html.Action* vloží akci *Create* z kontroleru komentářů, přes kterou se může vkládat nový komentář. Pod formulářem je vložené view *Comments*, kde jsou vypsané všechny komentáře ke článku, tak jak je předala proměnná *ViewData*.

Obrázek 5.15: Detail článku s odkazy na sociální sítě a možností vložení komentáře

5.11 Evidence klubů a hráčů

Veledůležitou součástí fotbalového webu jsou kluby a hráči. Na obě entity se zároveň vážou další. Ke klubu je přidružený stadion, klub patří do určité soutěže (kategorie) a země, země zase kontinentu. Hráč má kromě klubu přiřazenou také národnost. Modely

pro stadion, zemi a kontinent jsou jednoduché a obsahují pouze základní informace, kterou pak nese každá instance. Stadion má mezi atributy ID, název, adresu, kapacitu a případně obrázek. Země má kromě identifikačního čísla a názvu odkaz na kontinent a kontinent má pouze ID a název. Instance těchto entit může spravovat pouze administrátor webu.

Aby bylo možné do systému zadat klub, je potřeba mít vytvořenou zemi, ze které pochází, soutěž (kategorii), které se účastní a také stadion, na kterém hraje. Model klub má kromě těchto parametrů ještě další – název, rok založení, e-mail, telefon, adresu a dodatečnou textovou informaci. Návštěvníci webu mají možnost přidávat si kluby do seznamu oblíbených. Pro tento účel jsou v kontroleru vytvořené akce *Favorite* a *Unfavorite*.

```
public ActionResult Favorite(int clubId = 0)
{
      FavoriteClub favoriteClub = new FavoriteClub();
      favoriteClub.clubID = clubId;
      favoriteClub.userID =
(int)WebMatrix.WebData.WebSecurity.CurrentUserId;

      db.FavoriteClubs.Add(favoriteClub);
      db.SaveChanges();

      return RedirectToAction("Index", "Club");
}

public ActionResult Unfavorite(int clubId = 0)
{
      var actualId = (int)WebMatrix.WebData.WebSecurity.CurrentUserId;
      FavoriteClub favoriteClub = db.FavoriteClubs.Where(c => c.userID ==
actualId).Where(c => c.clubID == clubId).First();

      db.FavoriteClubs.Remove(favoriteClub);
      db.SaveChanges();

      return RedirectToAction("Index", "Club");
}
```

Zdrojový kód 5.14: Metody pro přidání a odebrání klubu z oblíbených

V obou metodách ve zdrojovém kódu 5.14 je vidět, že se pracuje s třídou *FavoriteClub*. Jedná se o třídu, která ve spolupráci s Entity Frameworkem eviduje v databázi informace o seznamu oblíbených klubů pro jednotlivé uživatele. Třída má pouze tři parametry – kromě svého ID také ID klubu a uživatele, který si ho přidal do oblíbených. Při vykonání obou akcí se předává ID klubu a ID uživatele se získává uvnitř akce z proměnné *WebMatrix.WebData.WebSecurity.CurrentUserId*, která jej uchovává.

V přidání oblíbeného klubu se vytvoří nová instance a uloží do databáze. Pokud se oblíbený klub odebírá, instance se vyhledá dle získaných parametrů a odstraní z databáze.

```
<% if (ViewBag.favorite == true) { %>
      <%: Html.ActionLink("Odebrat z oblíbených", "Unfavorite", new {
clubId=Model.id }, new{@onclick="return confirm('Odebrat klub z
oblíbených?')"}) %>
<% } else { %>
      <%: Html.ActionLink("Přidat do oblíbených", "Favorite", new {
clubId=Model.id }, new{@onclick="return confirm('Přidat klub do
oblíbených?')"}) %>

<% } %>
```

Zdrojový kód 5.15: Vypsání odkazů pro přidání nebo odebrání oblíbeného klubu

Zdrojový kód 5.15 zobrazuje odkazy pro přidání nebo odebrání oblíbeného klubu. O tom, jestli budeme klub odebírat nebo přidávat rozhoduje proměnná typu *boolean ViewBag.favorite*, která se nastavuje v kontroleru a předává do view:

```
public ActionResult Details(int id = 0)
{
      Club club = db.Clubs.Find(id);
      if (club == null)
      {
            return HttpNotFound();
      }

      ViewBag.favorite = false;
      var userId = (int)WebMatrix.WebData.WebSecurity.CurrentUserId;

      if (db.FavoriteClubs.Where(c => c.clubID == id).Where(c => c.userID
== userId).ToArray().Count() > 0)
      {
            ViewBag.favorite = true;
      }
      return View(club);
}
```

Zdrojový kód 5.16: Určení, zda je klub mezi oblíbenými před předáním jeho instance do view

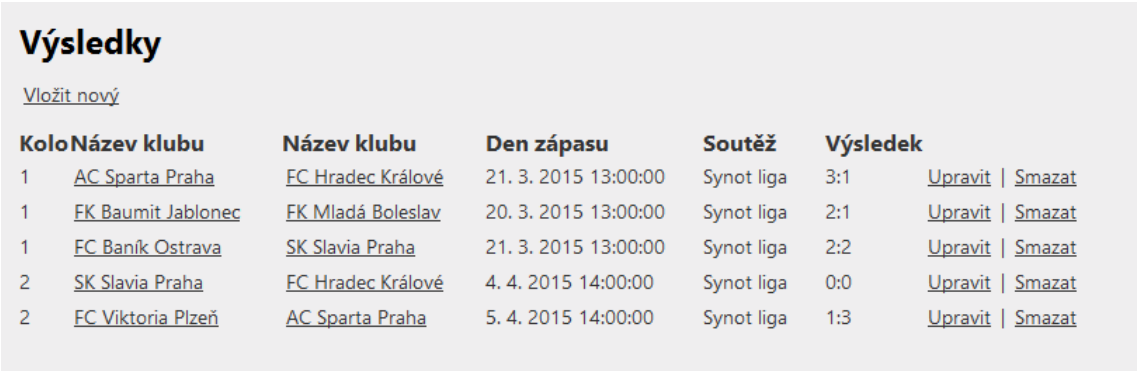
Kontroler vyhledá, zda již v databázi existuje instance třídy *FavoriteClub*, která by odpovídala ID klubu a uživateli, který si jej právě prohlíží. Pokud ano, nastaví proměnnou *ViewBag.favorite* na hodnotu *true*, v opačném případě zůstává v proměnné hodnota *false*.

Aby mohl administrátor do systému vložit hráče, musí být předem vložena země, ze které pochází a klub za který hraje. Další parametry hráče jsou ID, křestní jméno, příjmení, celé jméno, přezdívka, datum narození, váha, výška, číslo dresu a

fotografie. Nejzajímavější částí evidence hráčů jsou jejich statistiky, které budou popsány v následující kapitole.

5.12 Tabulky a hráčské statistiky

V systému je možnost v rámci soutěže zadávat výsledky, ze kterých se generuje tabulka soutěže. U každého výsledku lze zadat sestavu obou týmů a z těchto dat se pak počítají osobní statistiky hráčů.



Obrázek 5.16: Administrace výsledků soutěže

Obrázek 5.16 ilustruje view administrace výsledků v systému. Každý výsledek je zároveň i model s kontrolerem a tabulkou v databázi. Výsledek si nese informaci o ID zápasu, kole soutěže, domácím klubu, hostujícím klubu, datu a času zápasu, soutěži, sezóně (rok), počtu vstřelených branek pro domácí a počtu vstřelených branek pro hosty.

Tabulka soutěže

P	Tým	V	R	P	Skóre	Body
1	AC Sparta Praha	2	0	0	6:2	6
2	FK Baumit Jablonec	1	0	0	2:1	3
3	SK Slavia Praha	0	2	0	2:2	2
4	FC Hradec Králové	0	1	1	1:3	1
5	FC Baník Ostrava	0	1	0	2:2	1
6	FC Viktoria Plzeň	0	0	1	1:3	0
7	FK Mladá Boleslav	0	0	1	1:2	0
8	1. FC Slovácko	0	0	0	0:0	0
9	1. FK Příbram	0	0	0	0:0	0
10	Bohemians Praha 1905	0	0	0	0:0	0
11	FC Slovan Liberec	0	0	0	0:0	0
12	FC Vysočina Jihlava	0	0	0	0:0	0
13	FC Zbrojovka Brno	0	0	0	0:0	0
14	FK Dukla Praha	0	0	0	0:0	0
15	FK Teplice	0	0	0	0:0	0
16	SK Dynamo České Budějovice	0	0	0	0:0	0

Obrázek 5.17: tabulka soutěže vygenerovaná dle výsledků.

Ze všech zadaných výsledků soutěže se generuje aktuální tabulka na obrázku 5.17. Pro tento účel je v systému zavedená třída *TableResult*, jejíž instance se vytvářejí při generování tabulky a předávají do view *Table*:

```
public ActionResult Table(int categoryID)
{
	List<Club> clubs = db.Clubs.Where(c => c.categoryID == 
categoryID).ToList();
	List<TableResult> tableResult = new List<TableResult>();

	foreach (Club club in clubs) {

		TableResult clubResult = new TableResult();
		clubResult.clubId = club.id;
		clubResult.clubName = club.name;
		clubResult.wins = 0;
		clubResult.draws = 0;
		clubResult.losts = 0;
		clubResult.homeGoals = 0;
		clubResult.awayGoals = 0;
		clubResult.points = 0;
		clubResult.matches = 0;

		List<Result> results = db.Results.Where(r => r.homeClubID == 
	club.id || r.visitorClubID == club.id).ToList();

	foreach (Result result in results)
	{
		clubResult.matches++;
```

```
                    if (result.homeClubID == club.id) {
                          clubResult.points += result.homeClubPoints;
                          clubResult.homeGoals += result.homeClubGoals;
                          clubResult.awayGoals += result.visitorClubGoals;

                          if (result.homeClubGoals >
                    result.visitorClubGoals) {
                                clubResult.wins++;
                          }
                          else if (result.homeClubGoals <
                          result.visitorClubGoals) {
                                clubResult.losts++;
                          } else {
                                clubResult.draws++;
                          }
                    } else if (result.visitorClubID == club.id) {
                          clubResult.points += result.visitorClubPoints;
                          clubResult.homeGoals += result.visitorClubGoals;
                          clubResult.awayGoals += result.homeClubGoals;

                          if (result.homeClubGoals >
                    result.visitorClubGoals) {
                                clubResult.losts++;
                          }
                          else if (result.homeClubGoals <
                          result.visitorClubGoals) {
                                clubResult.wins++;
                          } else {
                                clubResult.draws++;
                          }
                    }
              }

              tableResult.Add(clubResult);
        }

ViewBag.clubs = tableResult.OrderByDescending(r =>
r.points).ThenByDescending(r => r.matches).ThenBy(r => r.clubName);

return View();


}
```

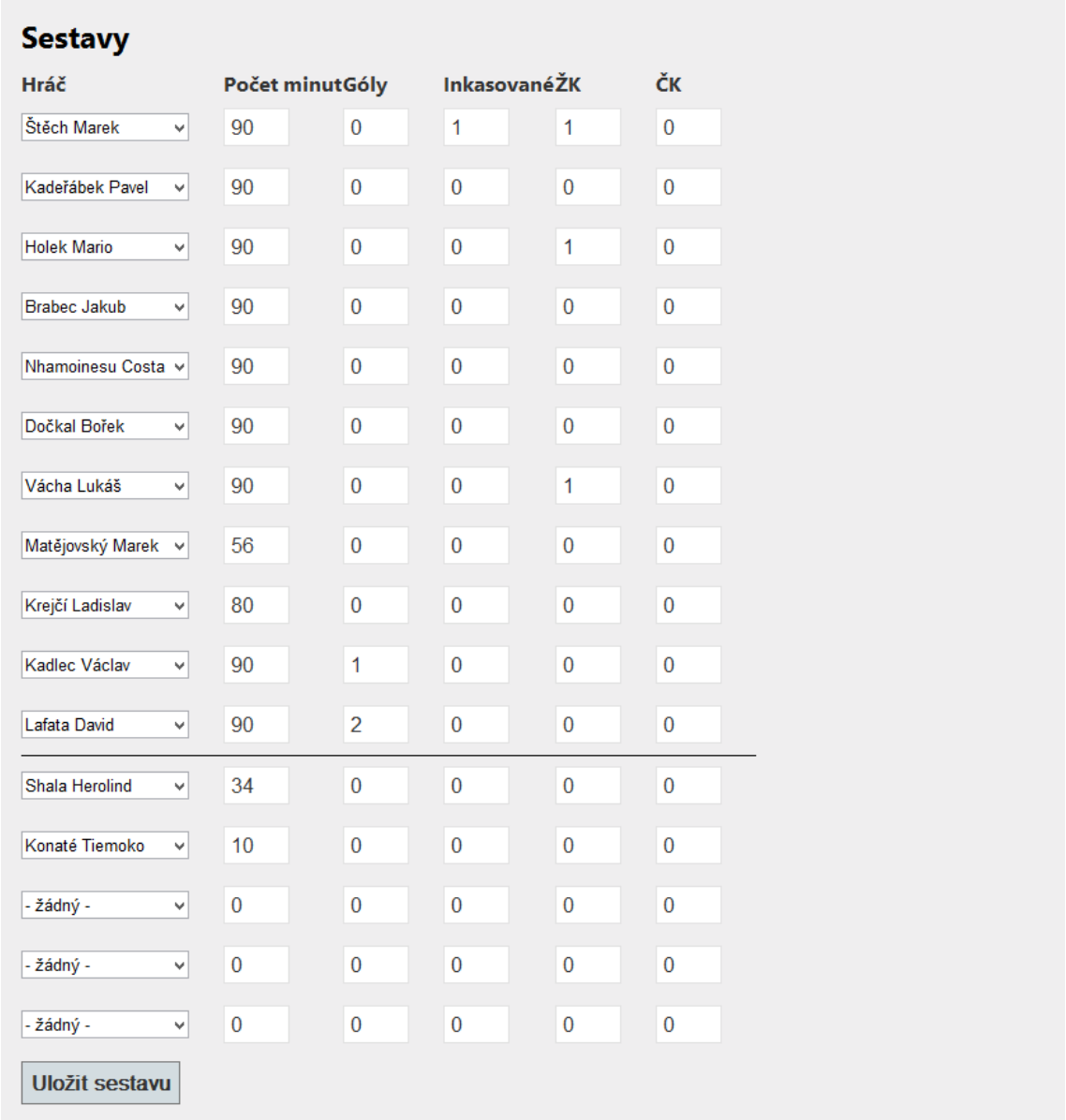
Zdrojový kód 5.17: Naplnění a předání tabulky soutěže v kontroleru

Algoritmus ve zdrojovém kódu 5.14 popisuje naplnění tabulky soutěže a předání do view. V parametru adresy se předává ID soutěže, ve které klub hraje. Jako první krok se tedy načtou všechny kluby ze soutěže a vytvoří se nový list *tableRestult*, do kterého se budou ukládat tabulkové výsledky klubů. Načtené kluby se pak pomocí *foreach* cyklu procházejí a pro každý z nich se vykoná stejná posloupnost úkonů. Nejprve se vytvoří nová instance třídy *TableResult* s názvem *clubResult*. U všech parametrů se uloží počáteční hodnoty. Název a ID klubu se předá z právě zpracovávaného klubu a u všeho ostatního (počet výher, remíz, proher, vstřelených branek, inkasovaných branek, bodů a

zápasů) se nastaví nula. Po této počáteční inicializaci klubového výsledku se dle ID klubu načtou všechny výsledky klubu do listu s názvem *results*. Výsledky klubu se hledají podle toho, že hledaný klub figuroval na pozici domácího nebo hostujícího týmu. Pro každý klub se v tomto místě začne vykonávat další *foreach* cyklus, který zpracuje všechny výsledky a doplní statistiky ke každému klubu. Ve funkci podmínka *if* rozlišuje, zda byl klub domácí nebo hostující. Tabulkové výsledky se v proměnné *ViewBag* předají do view, které vygeneruje tabulku:

```
@{
    ViewBag.Title = "Table";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Tabulka soutěže</h2>

<table>
    <tr>

<th>P</th><th>Tým</th><th>V</th><th>R</th><th>P</th><th>Skóre</th><th>Body
</th>
    </tr>

    @{var i = 1;}
    @foreach (var club in ViewBag.clubs)
    {
        <tr>
            <td>@i</td>
            <td>@club.clubName</td>
            <td>@club.wins</td>
            <td>@club.draws</td>
            <td>@club.losts</td>
            <td>@string.Format(club.homeGoals + ":"+ club.awayGoals)</td>
            <td>@club.points</td>
        </tr>
        i++;
    }

</table>
```

Zdrojový kód 5.18: View pro tabulku soutěže



Obrázek 5.18: Vložení sestav a hráčských statistik k zápasu

Jak ukazuje obrázek 5.18, ke každému výsledku lze vložit hráčské statistiky. Ty se vkládají vybráním hráčů, kteří se utkání zúčastnili a zapsáním informací o odehraných minutách, vstřelených nebo inkasovaných brankách a obdržených žlutých nebo červených kartách.

```
[Authorize(Roles = "admin, editor, redaktor, sefredaktor")]
public ActionResult Sheet(int id = 0, int clubID = 0)
{
      Result result = db.Results.Find(id);
      List<PlayerResult> playersResult = db.PlayerResults.Where(p =>
p.resultID == id && p.clubID == clubID).ToList();

      for (var i = playersResult.Count + 1; i <= 16; i++)
```

```
        {
              playersResult.Add(new PlayerResult { id = 0, resultID = id,
        playerID = 0, clubID = clubID, goals = 0, goalsReceived = 0, minutes
        = 0, redCards = 0, yellowCards = 0 });
        }

        ViewBag.playerID = new SelectList(db.Players.Where(p => p.clubID ==
clubID).OrderBy(p => p.lastName).ToList().Select(p => new SelectListItem {
Value = p.id.ToString(), Text = p.lastName + " " + p.firstName }),
"Value", "Text");

        return View(playersResult);
}
```

Zdrojový kód 5.19: Příprava view pro vložení hráčských statistik v akci kontroleru

Ve zdrojovém kódu 5.19 je ukázána příprava view pro vložení hráčských statistik. Akce *Sheet* získá z URL adresy ID utkání a klubu. Pro hráčské statistiky se používá třída *PlayerResult*, která ukládá informace o ID výsledku, hráče a jeho statistikách (branky, odehrané minuty, karetní napomenutí). Nejprve se tedy z databáze načtou již existující hráčské statistiky. Maximální počet hráčů, které můžeme zadat k jednomu zápasu je 16. Pokud ještě není k zápasu vložen maximální počet hráčů (nebo není vložený žádný), *for* cyklus doplní do listu, který se předává view, prázdné instance. Nakonec je do proměnné *ViewBag.playerID* načten seznam hráčů vybraného klubu, protože tato proměnná se ve view používá jako výběry hráčů do každého řádku.

```
@model List<FotbalPortal.Models.PlayerResult>

@{
    ViewBag.Title = "Sestavy zápasu";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Sestavy</h2>

<div>
    @using (Html.BeginForm("Sheet", "Result", FormMethod.Post)) {
        int j = 0;

        @Html.AntiForgeryToken();
        @Html.ValidationSummary(true);

        <table id="sheet">
            <tr>
                <th>Hráč</th><th>Počet
minut</th><th>Góly</th><th>Inkasované</th><th>ŽK</th><th>ČK</th>
            </tr>

        @foreach (var i in Model)
        {
            string trida = "";
            if (j == 10) { trida = "line"; }

            <tr class="@trida">
```

```
                <td>@Html.DropDownListFor(a => a[j].playerID, new
SelectList(ViewBag.playerID, "Value", "Text"), "- žádný -" )</td>
                <td>@Html.TextBoxFor(a => a[j].minutes)</td>
                <td>@Html.TextBoxFor(a => a[j].goals)</td>
                <td>@Html.TextBoxFor(a => a[j].goalsReceived)</td>
                <td>@Html.TextBoxFor(a => a[j].yellowCards)</td>
                <td>@Html.TextBoxFor(a => a[j].redCards)</td>
            </tr>
            @Html.HiddenFor(a => a[j].clubID)
            @Html.HiddenFor(a => a[j].resultID)
            @Html.HiddenFor(a => a[j].id)
            @Html.HiddenFor(a => a[j].playerID)

            j++;
        }

        </table>
        <input type="submit" value="Uložit sestavu" />
    }

</div>
```

Zdrojový kód 5.20: View pro sestavu týmu v zápase

Zdrojový kód 5.20 je view k akci *Sheet*. V modelu od kontroleru view obdrželo kolekci objektů typu *PlayerResult* a pomocí *foreach* cyklu sestaví celý formulář. Již dříve zmíněná proměnná *ViewBag.playerID* je zde předána jako výběr hráče metodou *Html.DropDownListFor*, která má tři parametry – konkrétní instanci přijatého modelu, *ViewBag.playerID* proměnnou předanou třídě *SelectList* a text pro nevybranou hodnotu. Posledním krokem je odeslání formuláře a zpracování dat v kontroleru:

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Sheet(List<PlayerResult> playersResult)
{
      if (ModelState.IsValid) {
            foreach (var player in playersResult) {
                  if (player.id == 0 && player.playerID > 0) {
                        db.PlayerResults.Add(player);
                  } else if (player.id > 0 && player.playerID > 0) {
                        db.Entry(player).State = EntityState.Modified;
                  } else if (player.id > 0 && player.playerID == 0) {
                        PlayerResult result =
                        db.PlayerResults.Find(player.id);
                        db.PlayerResults.Remove(result);
                  }
                  db.SaveChanges();
            }
            return RedirectToAction("Index");
      }

      ViewBag.playerID = new SelectList(db.Players.OrderBy(p =>
p.lastName).ToList().Select(p => new SelectListItem { Value =
p.id.ToString(), Text = p.lastName + " " + p.firstName }), "Value",
"Text");
      return View(playersResult);
```

```
}
```

Zdrojový kód 5.21: Kontrolerová akce pro sestavu týmu

Uložení všech informací spočívá v tom, že kontroler projde všechny instance, které mu vrátilo view. Pokud je ID záznamu 0 a ID hráče je vyšší než 0, je zřejmé, že se jedná o nově zadanou statistiku a informace se uloží do databáze. Když je ID záznamu vyšší než 0 a hráčské ID také, jedná se o již existující záznam, u kterého se mohly změnit údaje. Ten se označí jako modifikovaný a na konci cyklu je aktualizován v databázi. Poslední podmínka ošetřuje situaci, kdy je ID záznamu vyšší než 0, ale ID hráče je 0. Jedná se o upravený záznam, kdy byl hráč odstraněn a tím pádem je to i smazáno z databáze. Ve všech ostatních případech se jedná o prázdná pole, se kterými se nijak dále nepracuje. Na konci akce se znovu vyvolá view.

Z takto posbíraných dat je možné sestavovat hráčům jejich osobní statistiky. V systému budou dva pohledy na hráčské statistiky. Jeden bude v rámci týmové soupisky, kde bude celkový přehled všech hráčů. Druhý bude podrobnější, kde bude seznam zápasů, kterých se hráč účastnil a statistiky, kterých dosáhl.

Následující ukázka kódu je implementace soupisky v kontroleru klubu:

```
public ActionResult Team(int id = 0)
{
	Club club = db.Clubs.Find(id);
	if (club == null) {
		return HttpNotFound();
	}

	List<Player> players = db.Players.Where(p => p.clubID ==
	club.id).OrderBy(p => p.dressNumber).ToList();

	List<SquadPlayer> squad = new List<SquadPlayer>();

	foreach (Player player in players)
	{
		SquadPlayer _player = new SquadPlayer { playerID = player.id,
		name = player.lastName + " " + player.firstName, born =
		player.borned.ToString("dd.MM.yyyy"), matches = 0, minutes =
		0, goals = 0, number = player.dressNumber, redCards = 0,
		yellowCards = 0 };

		List<PlayerResult> playerResults = db.PlayerResults.Where(p =>
		p.playerID == player.id && p.clubID == club.id).ToList();

		foreach (PlayerResult result in playerResults) {
			_player.minutes += result.minutes;
			_player.goals += result.goals;
			_player.yellowCards += result.yellowCards;
			_player.redCards += result.redCards;
```

```
                    if (result.minutes > 0) {
                        _player.matches++;
                    }
                }

                squad.Add(_player);
            }

            ViewBag.squad = squad;

            return View();

}
```

Zdrojový kód 5.22: Kontrolerová akce pro soupisku týmu

Aby bylo možné do view předat vygenerovanou soupisku je potřeba mít vytvořenou další třídu pro načtení každého řádku. V tomto případě se jedná o třídu *SquadPlayer*, jejíž instance se předají do view. Třída uchovává ID hráče, jméno, datum narození, číslo dresu a informace o statistikách (počet zápasů, odehrané minuty, branky, žluté a červené karty). Na začátku akce s v kontroleru nejprve načtou z databáze všichni hráči konkrétního klubu seřazení vzestupně dle čísla dresu. Zároveň se deklaruje nový, zatím prázdný, list s názvem *squad* typu *SquadPlayer*. Poté se cyklem projdou všichni hráči klubu a postupně se u nich vykonávají stejné akce. Nejprve se vytvoří nová instance třídy *SquadPlayer*, která bude mít vynulované statistiky. Poté se z databáze načte seznam výsledků v listu typu *PlayerResult*, jejichž ukládání bylo popsáno již dříve v této podkapitole. Všechny výsledky se v dalším cyklu projdou a hráči se zaznamenají jeho statistiky. Na konci cyklu pro každého hráče se instance třídy *SquadPlayer* přidá do listu *squad*, který se po zpracování všech hráčů předá do view v proměnné *ViewBag*. Výsledek snažení je možné vidět na následujícím obrázku 5.19.

Soupiska

Č	Jméno	Narozen	Z	Min.	G	ŽK	ČK
1	Štěch Marek	28.01.1990	2	180	0	1	0
3	Kadlec Václav	20.05.1992	2	180	3	0	0
4	Švejdík Ondřej	03.12.1982	0	0	0	0	0
5	Brabec Jakub	06.08.1992	2	157	0	2	1
6	Vácha Lukáš	13.05.1989	2	180	0	1	0
8	Matějovský Marek	20.12.1981	2	126	0	0	0
9	Dočkal Bořek	30.09.1988	2	165	0	0	0
10	Shala Herolind	01.02.1992	1	34	0	0	0
11	Mareček Lukáš	17.04.1990	0	0	0	0	0
15	Kováč Radoslav	27.11.1979	1	20	0	0	0
16	Kadeřábek Pavel	25.04.1992	2	180	0	0	0
18	Konaté Tiemoko	03.03.1990	2	25	0	0	0
21	Lafata David	18.09.1981	2	180	3	1	0
22	Hušbauer Josef	16.03.1990	0	0	0	0	0
23	Krejčí Ladislav	05.07.1992	2	138	0	0	0
25	Holek Mario	28.10.1986	2	180	0	1	0
26	Nhamoinesu Costa	06.01.1986	2	180	0	0	0
35	Bičík David	04.06.1981	0	0	0	0	0
39	Podaný Jakub	15.06.1987	1	32	0	0	0

Obrázek 5.19: Vygenerovaná soupiska klubu s hráčskými statistikami

V klubové soupisce jsou shrnuty statistiky všech hráčů, v systému je ale také potřeba ukázat podrobnější údaje. Jméno každého hráče funguje zároveň také jako odkaz do přehledu s názvem *Stats*. Akce pro generování dat, která je vidět ve zdrojovém kódu 5.23, je umístěna v kontroleru pro hráče a do view předává seznam utkání, ve kterých hráč nastoupil včetně statistik.

```
public ActionResult Stats(int id = 0)
{
	Player player = db.Players.Find(id);
	if (player == null) {
		return HttpNotFound();
	}

	List<Result> results = db.Results.Include(r => r.homeClub).Include(r
	=> r.visitorClub).Where(r => r.homeClubID == player.clubID ||
	r.visitorClubID == player.clubID).OrderBy(r => r.round).ToList();

	List<PlayerStats> stats = new List<PlayerStats>();

	foreach (Result result in results) {
		PlayerStats _stats = new PlayerStats
```

```
            {
                  playerID = player.id,
                  resultID = result.id,
                  round = result.round,
                  goals = 0,
                  minutes = 0,
                  redCards = 0,
                  yellowCards = 0,
                  homeClub = result.homeClub.name,
                  visitorClub = result.visitorClub.name,
                  result = result.homeClubGoals + ":" +
result.visitorClubGoals,
                  date = result.date.ToString("dd.MM.yyyy")
            };

            var list = db.PlayerResults.Where(r => r.resultID == result.id
            && r.playerID == player.id).ToList();

            if (list.Count > 0) {
                  PlayerResult playerResult = list.First();

                  _stats.minutes = playerResult.minutes;
                  _stats.goals = playerResult.goals;
                  _stats.redCards = playerResult.redCards;
                  _stats.yellowCards = playerResult.yellowCards;
            }

            stats.Add(_stats);
      }

      ViewBag.stats = stats;
      ViewBag.playerName = player.firstName + " " + player.lastName;

      return View();
}
```

Zdrojový kód 5.23: Předání hráčských statistik do view

Proces zpracování dat probíhá velice podobně jako generování klubové soupisky. V tomto případě se zpracovává jeden hráč a na počátku se z databáze načtou všechny výsledky jeho klubu. Nenačítají se tedy pouze jeho uložené hráčské statistiky, protože v přehledu budou i zápasy, do kterých nezasáhl. Po načtení výsledků se založí prázdný list *stats* typu *PlayerStats*. Jedná se o třídu určenou pro zobrazení statistik v tomto view. Výsledky se pak v cyklu procházejí a pro každý se vytvoří nová instance třídy *PlayerStats*, do které se uloží kolo soutěže, datum zápasu, domácí a hostující klub a výsledek zápasu. Hráčské statistiky se nastaví na 0. Teprve poté se zjišťuje, zda pro tento zápas existuje v databázi statistika pro konkrétního hráče. Pokud ano, prázdné statistiky se nahradí a na konci cyklu se zpracovávaná instance přidá do listu *stats*. Výsledek generování detailní statistiky každého hráče je ukázán na následujícím obrázku.

Statistiky hráče Václav Kadlec

K	Datum	Zápas	Výsledek	Min.	G	ŽK	ČK
1	21.03.2015	AC Sparta Praha - FC Hradec Králové	3:1	90	1	0	0
2	05.04.2015	FC Viktoria Plzeň - AC Sparta Praha	1:3	90	2	0	0
Celkem				**180**	**3**	**0**	**0**

Obrázek 5.20: Detailní statistika hráče.

5.13 Videopříspěvky

Návštěvníci webu mají možnost nahrávat videopříspěvky a podělit se o ně s ostatními uživateli. Každé video nese informaci o tom, kdo ho kdy vytvořil. Na web se nenahrávají celé soubory, ale pouze odkaz na portál YouTube. Ve view se pak odkaz vkládá do vestavěného přehrávače. Zdrojový kód 5.24 je výpis videí vytvořený v zobrazovacím modulu ASPX.

```
<%@ Page Title="" Language="C#"
MasterPageFile="~/Views/Shared/Site.Master"
Inherits="System.Web.Mvc.ViewPage<IEnumerable<FotbalPortal.Models.Video>>"
%>

<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent"
runat="server">
    Videopříspěvky
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="MainContent"
runat="server">

<h2>Videopříspěvky</h2>

<% foreach (var item in Model) { %>
    <div>
        <h3><%: Html.DisplayFor(modelItem => item.name) %></h3>
        <%
            Uri myUri = new Uri(item.videoLink);
            string videoId =
HttpUtility.ParseQueryString(myUri.Query).Get("v");
        %>
        <iframe title="YouTube video player" class="youtube-player"
type="text/html"
        width="640" height="390" src="http://www.youtube.com/embed/<%:
Html.Raw(videoId) %>"
        frameborder="0" allowFullScreen></iframe>
    </div>
<% } %>

</asp:Content>
```

Zdrojový kód 5.24: View pro výpis videopříspěvků

Portál YouTube poskytuje vložený přehrávač jako prvek *iframe*, ve kterém se zobrazuje adresa *http://www.youtube.com/embed/ID_VIDEA*. Předává se tedy jen identifikační číslo videa, které je uvedené v úplné adrese. Například v adrese *https://www.youtube.com/watch?v=kieh4v-YjJY* je identifikační číslo videa *kieh4v-YjJY*. Tyto znaky je potřeba načíst z vloženého odkazu a použít pro vložený přehrávač. V kódu view se o to starají tyto řádky:

```
Uri myUri = new Uri(item.videoLink);
string videoId = HttpUtility.ParseQueryString(myUri.Query).Get("v");
```

Adresa se použije jako parametr pro vytvoření nové instance třídy *Uri*. Nad touto třídou je již možné provádět operace, které žádáme. V tomto případě se parametr *v*, který v adrese obsahuje ID videa rozparsuje do samostatné proměnné, kterou je možno použít do vestavěného přehrávače tímto způsobem:

```
<iframe title="YouTube video player" class="youtube-player"
type="text/html" width="640" height="390"
src=http://www.youtube.com/embed/<%: Html.Raw(videoId) %> frameborder="0"
allowFullScreen></iframe>
```

Parametr se do HTML kódu předává jako text metodou *Html.Raw* přímo do URL adresy videa.

Video má parametr *disabled*, který je nastaven jako výchozí hodnota při vložení návštěvníkem webu. Každé video musí schválit administrátor nebo editor a pro tento účel existuje v kontroleru pro videa akce *ApproveSet*:

```
<h2>Schválení videa</h2>

    <fieldset>
    <legend><%: Html.Raw(Model.name) %></legend>

    <%
       Uri myUri = new Uri(Model.videoLink);
       string videoId =
HttpUtility.ParseQueryString(myUri.Query).Get("v");
    %>
    <iframe title="YouTube video player" class="youtube-player"
type="text/html"
    width="640" height="390" src="http://www.youtube.com/embed/<%:
Html.Raw(videoId) %>"
    frameborder="0" allowFullScreen></iframe>

</fieldset>

<% using (Html.BeginForm()) { %>
    <%: Html.AntiForgeryToken() %>
    <%: Html.ValidationSummary(true) %>

    <fieldset>
```

```
        <legend>Video</legend>

        <%: Html.HiddenFor(model => model.id) %>
        <%: Html.HiddenFor(model => model.author) %>
        <%: Html.HiddenFor(model => model.videoLink) %>
        <%: Html.HiddenFor(model => model.created) %>

        <div class="editor-label">
            <%: Html.LabelFor(model => model.disabled) %>
        </div>
        <div class="editor-field">
            <%: Html.EditorFor(model => model.disabled) %>
            <%: Html.ValidationMessageFor(model => model.disabled) %>
        </div>

        <p>
            <input type="submit" value="Uložit" />
        </p>
    </fieldset>

<% } %>
```

Zdrojový kód 5.25: View pro schválení videopříspěvku

Schvalovatel si může na jednom místě přehrát video a následně je pod vestavěným přehrávačem formulář pro schválení. Po odeslání kontroler označí entitu jako upravenou a uloží změny v databázi:

```
private PortalContext db = new PortalContext();

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult ApproveSet(Video video)
{
       if (ModelState.IsValid)
        {
              db.Entry(video).State = EntityState.Modified;
              db.SaveChanges();
              return RedirectToAction("Approve");
        }
        return View(video);
}
```

Zdrojový kód 5.26: Schválení videopříspěvku v kontroleru

Po vykonání zdrojového kódu 5.26 je video schválené a objeví se mezi zveřejněnými.

5.14 Názory čtenářů

V rámci systémů je registrovaným návštěvníkům umožněno vyjadřovat svoje názory. Každý názor má kromě identifikačního čísla také autora, datum vytvoření a text. Názory návštěvníků se objevují na *Timeline* portálu, které je view vytvořené modulem Razor.

```
@model IEnumerable<FotbalPortal.Models.Opinion>

@{
    ViewBag.Title = "Názory";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Názory</h2>

<p>
    @Html.ActionLink("Vložit nový názor", "Create")
</p>

@foreach (var item in Model) {
    <div>
        <div class="author-date"><span>@Html.DisplayFor(modelItem =>
item.created)</span> - <span>@Html.DisplayFor(modelItem =>
item.author)</span></div>
        <p>@Html.DisplayFor(modelItem => item.text)</p>
    </div>

}
```

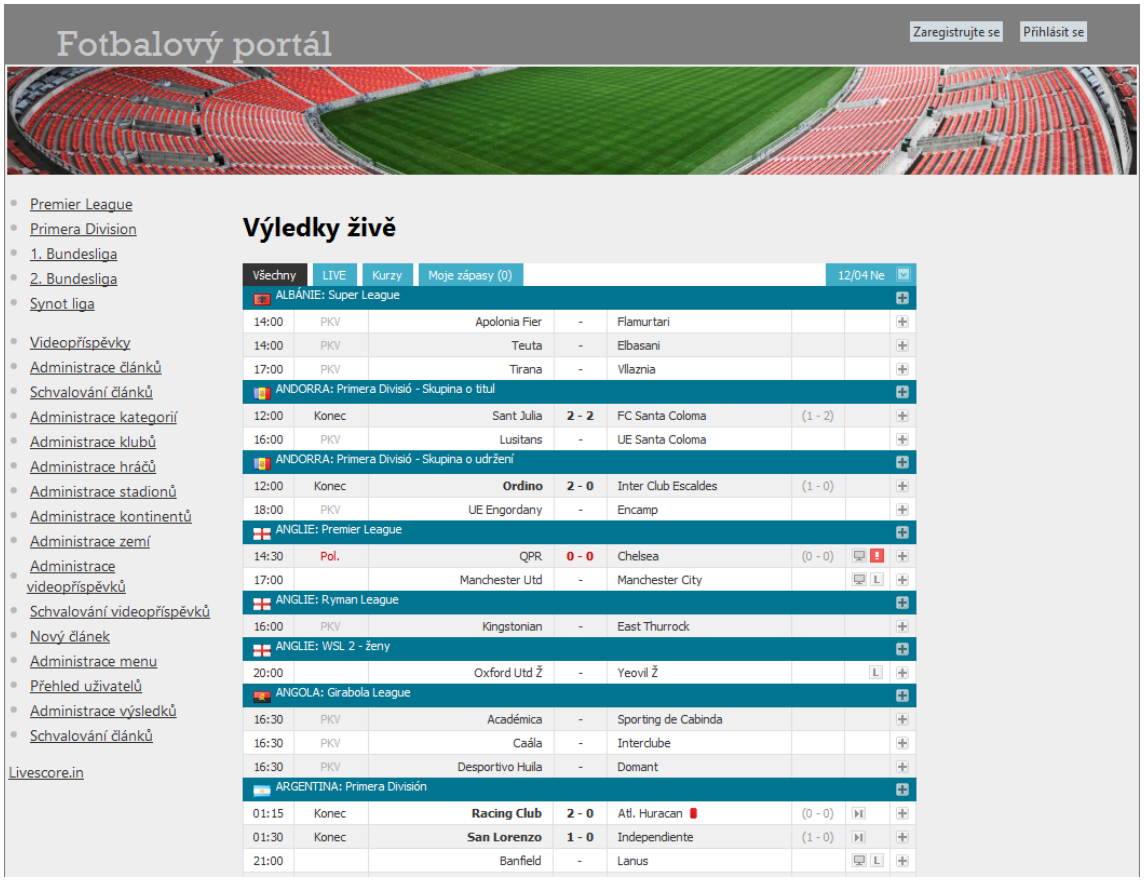
Zdrojový kód 5.27: View pro výpis názorů

První řádek ve zdrojovém kódu 5.27 deklaruje, že model, který se předává do view je vždy kolekce instancí třídy *Opinion* (názor). V další části Razor definuje titulek stránky a použitou šablonu layoutu webu. Dále je už HTML kód, do kterého zasahují dynamické části pro výpis všech názorů na stránce pomocí foreach cyklu.

5.15 Výsledky živě

Zpestřením pro návštěvníka webu a zároveň standardem jsou v dnešních dnech na sportovních portálech on-line aktualizované výsledky sportovních zápasů. Málokterý tým vývojářů a redaktorů má k dispozici prostředky na vlastní produkci výsledků, a tak je možné si na web vložit kód externí služby, která poskytne návštěvníkům výsledky za nás. V této webové aplikaci bude použita služba inScore (*http://www.livescore.in/cz/*). Poskytovatel služby nabízí bezplatné umístění výsledků administrátorům webů na jejich webové stránky pouze pod podmínkou vložení odkazu na domovský web.

Pro web v ASP.NET MVC tedy stačí vytvořit prázdné view s nadpisem a vložit vygenerovaný HTML/Javascriptový kód, který se o vše postará. V nabídce jsou výsledky z velkého množství zemí z celého světa s rozdělením na výsledky v rámci celého dne nebo zápasy, které se hrají živě. Výsledek vloženého kódu lze vidět na následujícím ilustračním obrázku 5.21.



Obrázek 5.21: On-line výsledky na webu

5.16 Nasazení

Pro již fungující projekt je nutné najít hosting, který podporuje používané technologie. Na českém trhu je spousta společností, které nabízejí hosting pro ASP.NET, ale už jen pouze dvě společnosti, které nabízejí hosting pro ASP.NET zdarma. Jedná se o hostingy ASPone (*http://www.aspone.cz/cz/*) a ASP2 (*https://asp2.cz/*). Z důvodů dobrých předchozích zkušeností si autor vybral prvně jmenovanou. Po registraci se celý projekt zkopíruje přes FTP protokol a databáze se vyexportuje do SQL dotazu. Projekt je následně dostupný na adrese *http://ehlousek.aspone.cz/*.

5.17 GIT

V dnešní době je již standardním postupem využívat při vývoji softwaru některý z dostupných systémů pro správu verzí. Tyto systémy jsou určené k tomu, aby zachovávaly historii všech změn, které se v souborech projektu po celou dobu uskutečnily. Tyto systémy nejčastěji používají vývojáři softwaru, protože díky nim mají kompletní přehled o tom, kdo kdy a kde upravil zdrojový kód aplikace. Týmům vývojářů tedy systémy pro správu verzí dopomáhají k lepší spolupráci a koordinaci práce a jednotlivcům nabízejí možnost zálohy jejich projektů s kompletní historií změn.

Práce s těmito systémy může probíhat dvěma způsoby. Existují systémy, kde jsou veškerá data centralizovaná na jednom serveru (*CVS*, *Subversion*) a každé uložení změny vyžaduje spojení se serverem. Na druhé straně jsou distribuované verzovací systémy (*Git*, *Mercurial*), ve kterých má každý vývojář lokálně uloženou kopii projektu. Na server se připojuje pouze v případě, že sám chce nahrát změny, což umožňuje rychlejší práci. Každá nahraná změna na server se nazývá revize a systém ji neuchovává celou, ale pouze rozdíl oproti té minulé a tím šetří velikost souborů.

Pro správu verzí tohoto projektu se použil sytém Git, se kterým má autor zkušenosti z jiných projektů. Git je systém vytvořený Linusem Torvaldsem, původně pro vývoj jádra Linuxu. Jedná se o svobodný software a jeho použití je velice jednoduché. Ovládání Gitu se provádí přes příkazový řádek, ale pro přátelštější použití je možné kompletní správu provádět přes některý z nástrojů, které Gitu poskytují uživatelské rozhraní (*SourceTree*, *SmartGit*, *GitHub*). Oproti ostatním systémům Git zpracovává data jiným způsobem. Již dříve bylo zmíněno, že verzovací systémy ukládají data jako rozdíly revizí. Git aplikuje jiný přístup, kdy chápe data jako sadu snímků (*snapshots*). Při každém uložení stavu projektu do systému udělá snímek toho, jak vypadají všechny soubory a uloží reference na tyto snímky. Pokud se v souborech nic nezměnilo, Git neukládá znovu celý soubor, ale pouze odkaz na předchozí identický soubor, který byl již dříve uložen. Není cílem této práce podrobně popisovat, jak celý systém Git funguje, ale aby byl některý projekt použitelný pro verzování, musí se v jeho adresáři vytvořit konfigurační soubory. Tyto konfigurační soubory obsahují informace, které Git potřebuje, aby s projektem správně pracoval. Kromě deklarace Gitu je zde i definice serveru, který hostuje soubory projektu.

Celý proces pak probíhá tak, že vývojář své změny lokálně uloží do revizí a poté spustí příkaz pro nahrání změn na server. Pokud mezitím jiný vývojář, který na projektu také pracuje, nahrál své změny, Git vyzve ke stáhnutí aktuální verze projektu a poté sloučí upravené soubory všech vývojářů projektu.

6 Zhodnocení

V této kapitole bude zhodnocena praktická část této práce. V tomto bodě je důležité podotknout, že autor před tvorbou projektu sice disponoval dostatkem zkušeností z oblasti webové tvorby, ale dosud neměl žádné praktické zkušenosti s použitím platformy .NET a její komponenty ASP.NET, o kterou tu výhradně šlo. Jednalo se o osobní zájem prostudovat tuto technologii a získat schopnosti k naprogramování webové aplikace.

Nejprve byl vytvořen ucelený návrh aplikace, který definoval jak technické, tak funkční požadavky webového portálu. Co se týče technických požadavků, byla použita moderní architekturu MVC s využitím objektově-relačně mapovacího databázového frameworku Entity Framework. Výsledné webové stránky byly generované dle nového HTML5 standardu. Následný vývoj byl tedy veden v souladu s moderními technologiemi a postupy.

Funkční požadavky na aplikaci se utvářely a upřesňovaly objektovým modelováním. Modelování případu užití pomohlo stanovit to, jaké budou uživatelské role a k čemu budou mít přístup. Zde je pozitivní, že se do určité míry povedlo implementovat vše, co bylo na počátku stanoveno. Webový portál se zpravodajskou tématikou a sociálním nádechem je poměrně velký projekt a na reálných projektech spolupracují početné týmy vývojářů, grafiků, redaktorů a vedoucích pracovníků. Ač to bylo přáním autora, nebylo v silách všechny případy užití doimplementovat do podoby, která by slušela webové aplikaci v ostrém provozu a s vysokou návštěvností. Výsledkem práce je ale solidní základ, který by mohl být při několika dalších vylepšeních připravený pro ostrý provoz.

Jak již bylo zmíněno, autor do projektu vstupoval s nulovými zkušenostmi s vývojem webových aplikací technologií ASP.NET. Přínosem práce je tedy také, že funguje jako průvodce pro vývojáře, kteří mají zkušenosti s webovým vývojem, ale chtějí proniknout do technologií vyspělejšího typu, jakou ASP.NET bezesporu je. Díky široké škále nástrojů nabízí velké možnosti ve vývoji nejen jednotlivcům, ale i týmům jakékoliv velikosti.

Během popisu implementace aplikace je demonstrováno jak využít moderní architekturu MVC a jak přenechat starosti o databázovou vrstvu mapovacímu frameworku. Součástí byla také implementace složitějších úkolů, jako jsou evidence výsledků fotbalových klubů a hráčských statistik, a jejich následná prezentace uživateli webu.

Kompletní zdrojové kódy a export databáze v SQL formátu se nachází na přiloženém CD.

7 Závěr

Závěrečná kapitola uzavře téma, kterému se v předchozích kapitolách věnovala tato diplomová práce. V první části byla pospána dlouhá historie technologie ASP.NET a její zajímavý vývoj. Jako součást .NET Frameworku přinesla webovým vývojářům nové možnosti a programátorům desktopových aplikací umožnila aplikovat známé postupy na webovou platformu.

V praktické části této práce je demonstrován návrh a implementace reálné webové aplikace. Pro implementaci byly zvoleny moderní technologie a postupy. Aplikace využívá architekturu MVC a pro práci s databází objektově-relačně mapovací Entity Framework. Popis implementace projektu je dle názoru autora vhodný pro webového vývojáře, který již má zkušenosti s programováním webových stránek, ale má zájem proniknout do komplexní technologie a naučit se základní principy vývoje v ASP.NET s použitím MVC architektury. Projekt je dopracován do funkční podoby, ale jako u každé webové aplikace, je zde prostor pro další vylepšení ať už po grafické stránce nebo po stránce uživatelského pohodlí.

V současné době se klade důraz především na rychlost vývoje, znovupoužitelnost programového kódu a proto se využívají různé frameworky pro skoro každý aspekt aplikace. Frameworky poskytují rozhraní různého rozsahu a starají se především o to, aby tvůrcům aplikací ulehčili práci a aby se mohli starat jen o ty zásadní věci při vývoji. Vývojář má na výběr širokou škálu nástrojů a je jen na něm, kterou cestu si vybere a co použije. ASP.NET je bez pochyby silná platforma, která je vhodná pro webové aplikace všech rozsahů. Může být použitelná pro rozsáhlé firemní systémy i pro docela malé prezentační webové stránky.

Lze předpokládat, že vzhledem k aktuálnímu trendu přesouvat vše důležité do cloudového prostředí, bude nadále sílit pozice webových aplikací ve světě informačních technologií. Tím pádem i technologie ASP.NET bude hrát nadále důležitou roli, což podporuje i společnost Microsoft tím, že vydává nové technologie a konstantně pracuje na nových verzích pro technologie stávající.

8 Použitá literatura

[1] MACDONALD, Matthew a Mario SZPUSZTA. *ASP.NET 2.0 a C#: tvorba dynamických stránek profesionálně*. Vyd. 1. Brno: Zoner Press, 2006, 1376 s. ISBN 80-868-1538-2.

[2] W3C. W3C [online]. 2012 [cit. 2014-03-01]. Dostupné z: http://www.w3.org/Consortium/

[3] What's New in the .NET Framework Version 3.5. Microsoft Developer Network [online]. 2012 [cit. 2015-04-18]. Dostupné z: https://msdn.microsoft.com/en-us/library/bb332048(v=vs.90).aspx

[4] LACKO, Luboslav. Vývoj aplikácií pre mobilené zariadenia [online]. Microsoft, 2000 [cit. 2014-03-01]. Dostupné z: http://msdn.microsoft.com/cs-cz/dd727769.aspx

[5] Preview .NET 2.0: Novinky v ASP.NET a ADO.NET. Weblog @ rebex.cz [online]. 2003 [cit. 2014-03-01]. Dostupné z: http://weblog.rebex.cz/blogs/bobr/archive/2003/09/22/151.aspx

[6] Serverové ovládací prvky Web Parts. MICROSOFT. Microsoft Developer Network [online]. 2014 [cit. 2014-03-01]. Dostupné z: http://msdn.microsoft.com/cs-cz/library/ms228267.aspx

[7] LÁSLO, Petr. Novinky v ASP.NET 3.5. Ajtaci.net [online]. 2009 [cit. 2014-03-01]. Dostupné z: http://ajtaci.net/programovani/net/aspnet/novinky-v-aspnet-35/

[8] LACKO, Ľuboslav. Ajax: hotová řešení. Vyd. 1. Překlad Michal Brůha. Brno: Computer Press, 2008. ISBN 978-80-251-2108-5.

[9] GARRETT, Jesse James. Ajax: A New Approach to Web Applications. Adaptive Path [online]. 2005 [cit. 2014-03-01]. Dostupné z: http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/

[10] HTTP/1.1 Pipelining FAQ. FISHER, Darin. Mozzila.org [online]. 2009 [cit. 2015-04-18]. Dostupné z: http://www-archive.mozilla.org/projects/netlib/http/pipelining-faq.html

[11] What's New in ASP.NET 4 and Visual Web Developer. Microsoft Developer Network [online]. 2011 [cit. 2015-04-18]. Dostupné z: https://msdn.microsoft.com/en-us/library/vstudio/s57a598e(v=vs.100).aspx

[12] Novinky v .NET Frameworku 4.5, 4.5.1 a 4.5.2. Microsoft Developer Network [online]. 2012 [cit. 2015-04-18]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/ms171868.aspx

[13] ASP.NET Tutorial. W3schools [online]. 2000 [cit. 2014-03-01]. Dostupné z: http://www.w3schools.com/aspnet/

[14] ASP.NET Web Pages - Tutorial. W3schools [online]. 2000 [cit. 2014-03-01]. Dostupné z: http://www.w3schools.com/aspnet/webpages_intro.asp

[15] ASP.NET Web Forms - Tutorial. W3schools [online]. 2000 [cit. 2014-03-01]. Dostupné z: http://www.w3schools.com/aspnet/aspnet_intro.asp

[16] ASP.NET WebForms – 1.díl – Úvod. Programujte.com [online]. 2012 [cit. 2014-03-01]. Dostupné z: http://programujte.com/clanek/2012033100-asp-net-webforms-1-dil-uvod/

[17] Introduction to ASP.NET and Web Forms. Microsoft Developer Network [online]. 2001 [cit. 2014-03-01]. Dostupné z: http://msdn.microsoft.com/en-us/library/ms973868.aspx

[18] ASP.NET MVC Tutorial. W3schools [online]. 2012 [cit. 2014-03-01]. Dostupné z: http://www.w3schools.com/aspnet/mvc_intro.asp

[19] ASP.NET MVC Overview. Microsoft Developer Network [online]. 2012 [cit. 2014-03-01]. Dostupné z: http://msdn.microsoft.com/en-us/library/dd381412(v=vs.98)

[20] Úvod do architektury MVC. Zdroják.cz [online]. 2009 [cit. 2014-03-01]. Dostupné z: http://www.zdrojak.cz/clanky/uvod-do-architektury-mvc/

[21] Prezentační vzory z rodiny MVC. Zdroják.cz [online]. 2009 [cit. 2014-03-01]. Dostupné z: http://www.zdrojak.cz/clanky/prezentacni-vzory-zrodiny-mvc/

[22] FREEMAN, Adam. Pro ASP.NET MVC 3 framework. 3rd. ed. New York: Apress, c2011, xxv, 824 s. The expert's Voice in.NET. ISBN 978-1-4302-3404-3.

[23] HTML5 Introduction. W3schools [online]. 2014 [cit. 2015-04-18]. Dostupné z: http://www.w3schools.com/html/html5_intro.asp

[24] CSS3 Introduction. W3schools [online]. 2014 [cit. 2015-04-18]. Dostupné z: http://www.w3schools.com/css/css3_intro.asp

[25] ARLOW, Jim a Ila NEUSTADT. UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky. Vyd. 1. Překlad Bogdan Kiszka. Brno: Computer Press, 2007, 567 s. ISBN 978-80-251-1503-9.

[26] JOHNSON, Bruce Ovid. Professional visual studio 2012. 1st ed. Indianapolis, IN: Wiley Pub., Inc., 2012, p. cm. ISBN 11-183-3770-0.

[27] MICROSOFT. Visual Studio [online]. 2015 [cit. 2015-04-18]. Dostupné z: https://www.visualstudio.com/

[28] ASP.NET Web Forms - Server Controls. W3schools [online]. 2010 [cit. 2015-04-18]. Dostupné z: http://www.w3schools.com/aspnet/aspnet_controls.asp

[29] ASP.NET Razor - Markup. W3schools [online]. 2012 [cit. 2015-04-18]. Dostupné z: http://www.w3schools.com/aspnet/razor_intro.asp

[30] Nástroje pro tvorbu UML diagramů. TIŠNOVSKÝ, Pavel. Root.cz [online]. 2005 [cit. 2015-04-18]. Dostupné z: http://www.root.cz/clanky/nastroje-pro-tvorbu-uml-diagramu/

[31] ADO.NET Entity Framework. Microsoft Developer Network [online]. 2009 [cit. 2015-04-18]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/bb399572.aspx

[32] Seznámení a instalace Microsoft SQL Serveru. JECHA, Tomáš. DotNetPortal.cz [online]. 2009 [cit. 2015-04-18]. Dostupné z: http://www.dotnetportal.cz/clanek/140/Seznameni-a-instalace-Microsoft-SQL-Serveru

[33] What's New in .NET Framework 4.5.1. DABADE, Pravinkumar. DotNetCurry [online]. 2014 [cit. 2015-04-18]. Dostupné z: http://www.dotnetcurry.com/showarticle.aspx?ID=968

[34] Novinky v sadě Visual Studio 2013. Microsoft Developer Network [online]. 2012 [cit. 2015-04-18]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/bb386063.aspx

[35] What is Visual Studio Online. Visual Studio [online]. 2014 [cit. 2015-04-18]. Dostupné z: https://www.visualstudio.com/en-us/products/what-is-visual-studio-online-vs.aspx

[36] Moving Java Applications to .NET. Microsoft Developer Network [online]. 2005 [cit. 2015-04-22]. Dostupné z: https://msdn.microsoft.com/en-us/library/ms973842.aspx

[37] HTTP Pipelining: A security risk without real performance benefits. MACVITTIE, Lori. [online]. 2009 [cit. 2015-04-22]. Dostupné z: https://devcentral.f5.com/articles/http-pipelining-a-security-risk-without-real-performance-benefits

[38] Entity Framework Overview. Microsoft Developer Network [online]. 2012 [cit. 2015-04-22]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/bb399567(v=vs.110).aspx

[39] Web Sockets. MALÝ, Martin. Zdroják.cz [online]. 2009 [cit. 2015-04-24]. Dostupné z: http://www.zdrojak.cz/clanky/web-sockets/

Příloha 1 – Obsah CD

Přílohou k diplomové práci je CD, které obsahuje praktický projekt. V adresáři *FotbalPortal* se nachází projekt, který lze otevřít ve Visual Studiu 2012 nebo 2013. Vedle této složky je také export databáze v souboru *Database.sql*. Návod pro zprovoznění projektu na lokálním serveru je popsán v souboru *README.txt*.