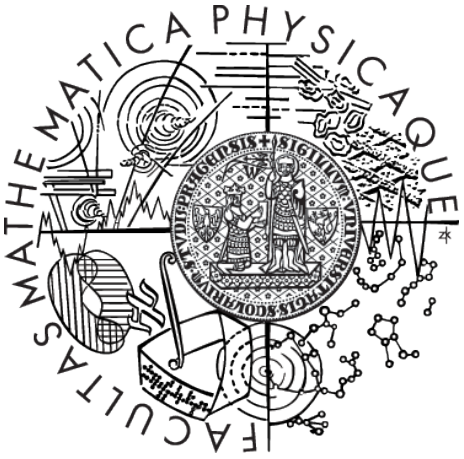
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Václav Fajta

Webová aplikace pro sportovní ligy

Katedra softwarového inženýrství

Vedoucí bakalářské práce: Mgr. Ladislav Maršík
Studijní program: Informatika
Studijní obor: programování

Praha 2015

Především bych rád poděkoval mému vedoucímu bakalářské práce Mgr. Ladislavu Maršíkovi za rady, připomínky a pomoc nejen s prací samotnou, za flexibilitu a ochotu najít si na mě čas i přes svůj nabitý program, za představení mnoha nástrojů, jež mě budou sloužit po celý život, za zjištění i hlídání termínů, rychlé reakce a důkladný testing, u kterého jistě strávil mnoho času a v neposlední řadě za jeho trpělivost, ochotu, podporu a přátelský přístup. Lepšího vedoucího bych hledat jen stěží.

Nemalé díky patří rodině za jejich podporu, bez které bych nedošel až sem.

Děkuji i Tomáši Součkovi za jeho bleskově rychlé připomínky a návrhy na zlepšení.

V neposlední řadě děkuji i hráčům hradecké squashové ligy za odzkoušení práce v reálném prostředí a zajištění, aby práce vycházela z opravdových potřeb hráčů.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Název práce: Webová aplikace pro sportovní ligy

Autor: Václav Fajta

Katedra: Katedra softwarového inženýrství

Vedoucí bakalářské práce: Mgr. Ladislav Maršík, Katedra softwarového inženýrství

Abstrakt: Organizace ligových soutěží jednotlivců ve sportovních centrech přináší různá úskalí. Sportovci musí mít přístup k výsledkům a statistikám, organizátoři ligy kromě kontroly průběhu potřebují nastavovat a měnit pravidla, způsoby vyhodnocováni a organizovat turnaje. Cílem této práce je vývoj aplikace, která by takovou činnost usnadnila. Zavádíme proto přizpůsobitelný systém, kde je možné spravovat více lig s rozdílnými nároky pod jedinou webovou aplikací. Vypořádá se s vícero sporty, různými systémy zápasů i specifickými pravidly každé soutěže. Postavena je na platformě ASP.NET a návrhovém vzoru Model-View-Controller. V práci popisujeme technologie a architekturu vývoje webových aplikací a vysvětlujeme princip práce našeho systému na diagramech a ukázkách.

Klíčová slova: webová aplikace, administrační software, ligové sporty, turnajový software, model view controller

Title: Web application for sports leagues

Author: Václav Fajta

Department: Department of Software Engineering

Supervisor: Mgr. Ladislav Maršík, Department of Software Engineering

Abstract: Organization of individual league competitions in sport centres brings various difficulties. Players need to have access to their results and standings, organizers besides monitoring the league need to set the rules, evaluate the results and organize tournaments. The aim of this work is development of an application which would make such activity simpler. We bring adjustable system, where it is possible to manage a lot of leagues with different demands under one web application. It handles multiple sports, diverse match systems and specific rules of each league. It is built on ASP.NET platform using Model-View-Controller design pattern. We describe technologies and architecture for web development, and present our application using diagrams and screenshots.

Keywords: web application, administration software, sports leagues, tournament software, model view controller

Obsah

1. Úvod

S narůstajícím zájmem o sporty, jako je squash a ricochet, přibývá stále více sportovních center umožňujících tyto sporty provozovat, často i s mnoha dalšími a pod jednou střechou. Mnozí z nich brzo zjistí, že dobrým způsobem jak ještě zvětšit zájem o daný sport a zvýšit si tak svou klientelu je založení místní ligy.

Liga zajišťuje pravidelný přísun zápasů i soupeřů. Soutěžní charakter navíc poskytuje motivaci ke zlepšení. Nejeden hráč se tak vybičuje k lepším výsledkům a bojuje i při zbytku sil déle a usilovněji, než jak by tomu bylo při běžném přátelském utkání. Díky rozdělení dle výkonnosti se každý po nějaké době ustálí okolo své úrovně, a zápasy proto budou vyrovnané a napínavé. Účastníkovi tak odpadá starost o shánění soupeřů, což není vždy lehké, obzvláště mají-li být na podobné úrovni, jako je sám. Potká se s různými styly hraní a získá tak zkušenosti, které by jinak musel shánět se značným úsilím. I přirozenou lenost spíše překoná, má-li jít hrát zápas do soutěže, než kdyby šel již posté s tím samým člověkem, u kterého již předem dokáže s poměrně velkou jistotou říci, jak to dopadne. Sledováním pohybu ve skupinách může relativně jednoduše vyhodnotit zlepšení či zhoršení své hry a momentální formu.

Pro centrum to znamená pravidelně obsazené kurty a celkové zvyšování úrovně sportu, jež může vyústit až v založení vlastního republikového družstva a díky tomu k prezentaci svého sportovního zařízení i v jiných městech.

Založení ligy se sebou však přináší organizační problémy. Ze začátku si asi vystačí s tisknutím a rozvěšováním informací po nástěnkách. Když však počet účastníků narůstá, což je koneckonců účelem, začíná složitost rychle stoupat. Například jak zajistit distribuci kontaktů na hráče, aby si mezi sebou mohli domlouvat časy zápasů? Nutné je i to, aby byly známé veškeré výsledky zápasů. I samotné vyhodnocení skupin zabírá také stále více času. Centrum brzy přijde na to, že bude třeba pro tyto účely zajistit nějaký program.

A právě k organizaci takové soutěže má tato aplikace sloužit. V jednoduchosti je síla, proto by ovládání nemělo být složité. Přesto by měla být zachována jistá míra obecnosti kvůli přizpůsobení se. Přece jen je pohodlnější mít jednu aplikaci i v centrech, kde mají kurty squashové i badmintonové a provozují ligy u obou sportů. Squash se typicky hraje na tři vítězné sety, zatímco badminton na dva vítězné. Nebo souběžně probíhají ligy dospělých a juniorů. Aplikace to má zohlednit. Řešit daný problém na moc velké úrovni obecnosti také není žádoucí, aby se obsluha nestala příliš složitou.

Důležitým aspektem je i jazyk. Soutěže se mohou účastnit hráči každého věku, od nejmladších až po veterány. Pro mnoho z nich je angličtina stále nepřekonatelný problém. Nemělo by se ani zapomenout, že v dnešní době jsou smartphony i tablety opravdu běžné, proto je více než vhodné, aby se uživatelské rozhraní dovedlo takovým zařízením přizpůsobit.

Zatím jsme si představili pouze soutěže jednotlivců, u nich však také zůstaneme. Pro družstva je třeba větší počet hráčů, než centrum typicky má. Zápasy by kvůli rozdílným časovým možnostem hráčů nemohli probíhat tak často. Výhody pro centrum tak zmizí. Družstevní soutěže individuálních sportů samozřejmě existují, ale alespoň u nás se hrají spíše pod hlavičkou nějakého svazu na republikové úrovní. To již zdaleka přerůstá centrum, pro které je aplikace určena. Jejich

požadavky jsou velice rozdílné a spojení obou konceptů dohromady by přineslo různé složitosti, o které ani jeden z nich nestojí. Každému to své. Tato práce sa věnuje vytvoření aplikace pro sportovní centra pořádající soutěže jednotlivců. Inspirovaná je především prací squashových a badmintonových center.

1.1 Struktura práce

V kapitole 2 si představíme některé práce zabývající se podobnou tématikou a v kapitole 3 podíváme se na technologie, na kterých je aplikace postavena. V kapitole 4 se budeme zabývat architekturou, tedy jak je aplikace postavena. Aplikaci samotnou, jak se člení, s jakými problémy se musí vypořádat, jaké používá modely a algoritmy si předvedeme v kapitole 5 a zakončíme to v kapitole 6 pohledem ze strany uživatele a předvedením některých zajímavých funkcí.

1.2 Liga jednotlivců

Pro přehlednost bychom měli uvést, jak taková typická liga jednotlivců vypadá. Jde o dlouhodobou soutěž. Jeden ročník, či sezóna chcete-li, trvá většinou o něco méně než jeden rok. Ročník je rozdělen na kola, v každém kole jsou hráči rozděleni do skupin dle svých výsledků v kole předchozím. Ve skupině se hraje stylem každý s každým. Podle různých kritérií se na konci kola hráči ve skupinách seřadí, nejlepší z nich postupují do lepších skupin, nejhorší sestupují níže. Za výsledky zápasů a za účast ve skupině získávají hráči body. V průběhu ročníku se mohou konat i turnaje, za které se také udělují body. Vítězem pak bývá hráč s největším počtem nasbíraných bodů.

1.3 O autorovi

Ne málo poznatků, které byly při tvorbě tohoto projektu brány v potaz, autor čerpal z vlastní zkušenosti. Sám jednu takovou ligu mnoho let vede, byl při vzniku další a navštívil, ať už jako divák či hráč, mnoho turnajů a velkou řadu z nich i sám organizoval. Byl u malých či větších turnajů lokální úrovně, turnajů úrovně republikové, družstevních, u mnoha mistrovství republiky a i u některých mezinárodních akcí jako třeba mistrovství Evropy juniorů a klubů. Radil se i s mnoha kolegy vedoucí ligy jiné.

2. Podobné práce

V této kapitole si představíme jiné práce podobné naší. V době přípravy a vzniku tohoto projektu jsme však neobjevili žádnou volně dostupnou aplikaci zabývající se stejným problémem na stejné úrovni. Tím ovšem motivace k tomuto dílu jen vzrostla.

Různá sportovní centra mají většinou nějakou svojí aplikaci, která jim problematiku ligy řeší. Ty, ke kterým jsme měli možnost se dostat, však trpěli stejným problémem. Mnohdy vznikali v rychlosti na koleni, aby liga měla s čím pracovat. Už z ovládání je tak znát pro jaký sport a ligu vznikali. Prostor pro přizpůsobení nebývá. Stavěny jsou pro konkrétní systém jediné ligy. A správcem ligy by ideálně měl být rovnou správce systému, protože je pro mnoho úkonů nezbytný.

Autor měl nějakou dobu možnost pracovat v jedné takové aplikaci pro ligové hraní. Jde o typickou ukázku, co se v centrech nachází. A rozhodně jí patří dík. Několik dlouhých let centru sloužila a stala se i podmětem ke vzniku tohoto projektu. Byla i jistým vzorem, jak by se některé části měli provést či naopak neměli. Tato aplikace má následující důležité nedostatky:

- Umožňuje provozovat jedinou ligu.
- Používá jediný napevno stanovený systém zápasů.
- Pravidla pro rozlosování kol a utváření skupin jsou neměnná. Při změně je tak nutno nechat aplikaci upravit.
- Přestože poskytuje nějaké statistiky, neumožňuje výběr dat z čeho se mají počítat, čímž značně ztrácí vypovídající hodnotu.
- Aplikaci chybí moderní uživatelské rozhraní.
- Administrátorské možnosti jsou chudé. Správce tak například musí každý nepotvrzený zápas potvrdit ručně, což při stovkách zápasů zabere opravdu hodně času.
- Uživatelské rozhraní není user-friendly a intuitivní. Uživatel si tak některých funkcí vůbec nevšimne a dostat se k datům může být velký problém.
- Uživatelé ani nepotěší to, že kontakty na ně jsou dostupné všem uživatelům aplikace i potom co ligu opustili. Malinko to kompenzuje předchozí bod. Dostat se ke kontaktům jiných hráčů, než se kterými je zrovna ve skupině, je natolik neintuitivní, že si toho uživatel často vůbec nevšimne.

Čtenář mohl slyšel o TournamentSoftware[1]. Ten se, jak název napovídá, soustředí na organizaci turnajů a to na mezinárodní úrovni. Podporuje mnoho sportů včetně těch kolektivních. S jeho pomocí se uskutečnění turnajů ve sportech jako squash, fotbal či třeba i šipky stane jednodušší. Použít lze více systémů jak hrát. Nechybí přehled organizačních informací jako kdy a kde se turnaj koná. Poskytuje časový plán zápasů včetně centra/hřiště/kurtu, kde mají probíhat. Prohlédnout

[1] http://tournamentsoftware.com/

si je možné soupisky týmů/hráčů jak dle abecedy, tak dle země, ze které pochází. Nechybí ani přehled s procentuální úspěšností vybraného týmu/hráče nebo třeba žebříček, pokud hráči získávají za umístění body. Přestože umožňuje výběr z několika jazyků uživatelského rozhraní, český mezi nimi v době psaní práce nebyl. Rozhodně je to šikovný program, byť míří na jinou cílovou skupinu.

V přibližně stejné době jako byla vyvýjena naše aplikace, vznikal ve Spojených státech program SportyHQ[2]. Jeho cíle jsou podobnější těm našim, zahrnuje ale i mnoho dalších oblastí, se všemi výhodami i nevýhodami, které to přináší. Opět jde spíše o národní až mezinárodní úroveň. A zajímá se také o kolektivní sporty. Poskytuje nástroje pro správu a propagaci lig a turnajů. Samozřejmostí jsou i statistiky. Nesnaží se ale být jen sportovní aplikací. Jeho cílem je stát se i komunitou a sociální sítí. Je to zároveň i rezervační systém. Myšlenka je to slibná. V době psaní práce jsme neobjevili žádnou funkci pro změnu jazyka z angličtiny na jiný, což ale není překvapivé, vhledem k tomu, že byl mířen pro americký trh.

[2]http://www.sportyhq.com/

3. Použité technologie

V této kapitole si představíme technologie, o které se aplikace opírá.

3.1 .NET Framework

Celá aplikace je postavena na platformě .NET Framework vyvíjené firmou Microsoft. Ten je navrhnut tak, aby podporoval interoperabilitu starých a nových programů, jazykovou neutrálnost, přenositelnost, bezpečnost a prostředím řešenou správu paměti.

Zjednodušený pohled na .NET je znázorněn na obrázku 3.1. V přehledu nejsou zahrnuty ani všechny verze, ani všechny komponenty. Framework obsahuje mnoho dalších komponent (jako WPF[1],WCF[2],...), které ale pro naše účely nejsou podstatné. Jednotlivé části .NET Frameworku znázorněné na obrázku si představíme v podkapitolách.

3.1.1 Principy .NET

CLI Common Language Infrastructure (CLI) je specifikace popisující spustitelný kód a prostředí, ve kterém běží. Definuje platformově a jazykově nezávislé prostředí, společnou množinu datových typů, metadata pro popis programů i jak se ošetřují výjimky. Udává základní pravidla pro CLI kompatibilní jazyky, jak mají být programy nahrávány a spouštěny. Díky tomu je možno psát programy pro .NET v různých jazycích.

CIL Zdrojové kódy nejsou překládány rovnou do strojového kódu, nýbrž do mezijazyka tzv. bytecode. Mezijazyk pro CLI se nazývá Common Intermediate Language (CIL). Přeložený program do CIL je uložen v CIL assembly, kterou pak můžeme použít na různých platformách. Odpadá tak nutnost mít kód přeložený několikrát dle toho, na které platformě má běžet. Vystačíme si s jedinou assembly, viz obrázek 3.2

CLR .NET aplikace se spouští a běží v softwarovém prostředí nazývaném Common Language Runtime (CLR). To je již platformě závislá implementace CLI. Jedná se o virtuální stroj, ne nepodobný java virtual machine, zajišťující běhové prostředí pro .NET programy. Přeloží assembly do instrukcí strojového kódu, který pak může běžet na uživatelově procesoru.

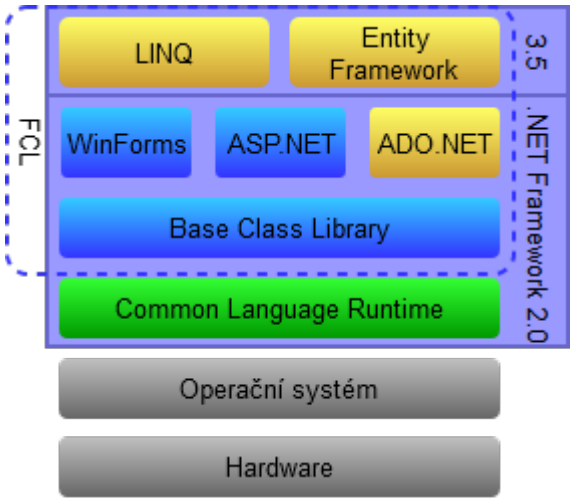
Je tedy jedno v jakém jazyce byl program napsán, spouštěn je pomocí CLR. Jazyky pro .NET jsou označovány jako „managed“ právě proto, že pro svůj běh potřebují runtime prostředí, na rozdíl od native jazyků, jež ho nepotřebují. Dalším příkladem managed jazyka je java.

CLR navíc poskytuje další služby jako správu paměti, zpracování výjimek, typovou bezpečnost, správu vláken, zabezpečení a další.

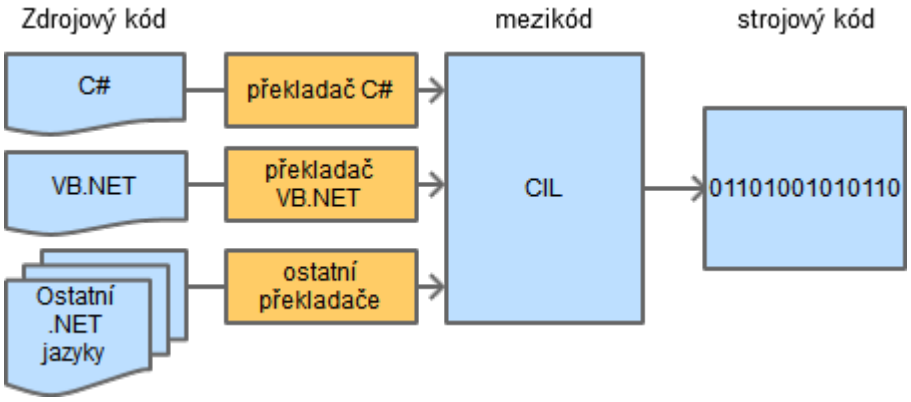
CLR je zahrnuta v každé verzi .NET frameworku.

[1]Windows Presentation Foundation; platforma pro vytváření bohatého uživatelského rozhraní

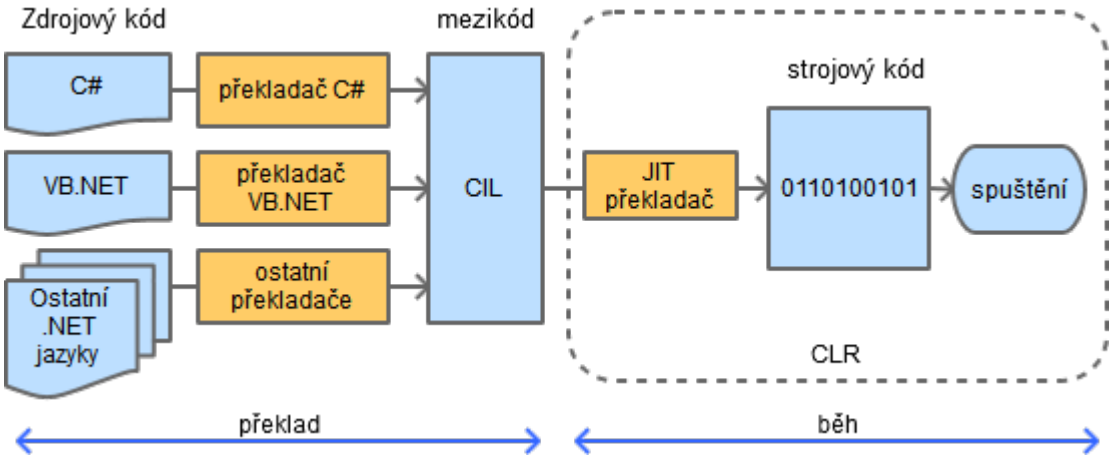
[2]Windows Communication Foundation; pro propojené servisně orientované aplikace



Obrázek 3.1: Zjednodušený pohled na .NET Framework



Obrázek 3.2: .NET jazyky jako je C# a Visual Basic jsou překládány do mezikódu



Obrázek 3.3: Překlad .NET jazyků do strojového kódu probíhá až za běhu místo při kompilaci

JIT překlad Překlad CIL assembly probíhá Just-In-Time (JIT). To znamená, že je přeložena do strojového kódu až při běhu, když je potřeba, a ne již při kompilaci. Výhodou pak je například to, že daný kód může být lépe optimalizován pro procesor, na kterém bude spouštěn. Nevýhodou je ale počáteční prodleva. I jednotlivé části jsou překládány, až když jsou třeba. Pokud tedy nějaký kód není použit, nemusí být vůbec přeložen. Proces je znázorněn na obrázku 3.3, na jeho vytvoření byly použity informace z knihy Beginning ASP.NET [7].

Stojíme-li o výhody klasického překladu, můžeme použít nástroj Native Image Generator[3] pro klasickou kompilaci.

3.1.2 Standardní knihovny pro .NET Framework

V .NET Frameworku jsou zahrnuty i standardní knihovny tříd. Ty jsou organizovány pomocí hierarchie jmenných prostorů.

Základní množinu tříd tvoří Base Class Library (BCL). Obsahuje základní typy, třídy pro práci se soubory, kolekce, práci s řetězci a pod.

Nadmnožinou je pak Framework Class Library (FCL) obsahující třídy pro tvorbu webových aplikací, aplikací s grafickým rozhraním, práci s daty, kreslení a mnohé další.

3.2 ADO.NET

ADO.NET je součást .NET Frameworku zabývající se službami pro přístup k datům. Přičemž se nemusí jednat pouze o data uložená v relačních databázích.

ADO.NET je zaměřen mimo jiné na zjednodušení programátorovy práce při manipulaci s daty[4]. Poskytuje pro to silně typovanou syntax. Snižuje velikost kódu potřebnou k provedení datového úkonu. Snaží se redukovat množství problémů vznikajících při spojení relačních databázových systémů s programy psanými v objektově orientovaných jazycích [5].

[3] Je dodáván spolu s frameworkem.

[4] Tento trend přichází s verzí 3.5 [5]

LINQtoObjects	Základní implementace umožňující dotazy nad kolekcemi implementující `IEnumerable<T>` nebo `IQueryable<T>`
LINQtoSQL	Dotazování nad Microsoft SQL Server databázemi (včetně SQL Compact)
LINQtoXML	Dotazování nad XML dokumenty
ParallelLINQ	Rozšíření LINQtoObject umožňující paralelní zpracování dotazů na vícejádrových procesorech.
LINQtoAmazon	Pro provádění dotazů nad knihami z Amazonu, implementoval Fabrice Marguerie [11]

Tabulka 3.1: Různé implementace LINQ vybrané z knihy Professional ADO.NET [5]

Programátor používá podobný kód pro přístup k datům, ať už dělá aplikaci webovou, desktopovou klient/server či jednouživatelovou.

3.3 LINQ

Od verze 3.5 je součástí .NET Frameworku i Language Integrated Query (LINQ). LINQ definuje .NET API a přidává novou syntax do jazyků C# a VB.NET, umožňující provádět dotazy nad různými datovými typy. Syntax LINQ se podobá SQL a je silně typovaná.

Podporované datové typy je možno rozšiřovat [14]. Vytvořit příslušný provider není navíc vůbec obtížné [5]. Programátorovi se tak do ruky dostává mocný nástroj. Místo aby se musel učit nové jazyky a dialekty (SQL, eSQL, LDAP,...) vystačí si s jediným, a navíc silně typovaným jazykem. V tabulce 3.1 můžeme vidět některé vybrané implementace LINQ.

Provádět dotazy je možné téměř nad jakýmkoliv zdrojem informací, který implementuje rozhraní `IEnumerable<T>` či `IQueryable<T>`.

V LINQ programátor říká, co chce provést místo jak. Patří tedy stejně jako SQL mezi deklarativní jazyky.

3.4 Entity Framework

Entity Framework (EF) pro objektově relační mapování je také součástí .NET Frameworku od verze 3.5. Umožňuje pracovat s relačními daty jako s doménově specifickými objekty. Výhodou opět je, že programátor pracuje se silně typovanými objekty.

Přičemž je jedno, zda databáze existuje předem, nebo zda jí budeme teprve vytvářet. Můžeme si nechat vygenerovat třídy podle již existující databáze, nebo naopak vytvořit třídy a dle nich si nechat databázi vytvořit. Anebo si můžeme namodelovat schéma databáze pomocí designeru a nechat si vytvořit jak databázi, tak třídy.

To nám umožňuje použít „Code-First“ přístup, čehož využijeme. Vytvoříme si třídy odpovídající objektům našeho problému a přidáme k nim datové anotace pomocí atributů (jako Key, Required a pod.). K popisu je možné využít i Fluent API. Databáze se nám vygeneruje sama dle typů proměnných a anotací, aniž

bychom museli cokoliv modelovat nebo psát příkazy SQL. Pomocí migrace si poradíme i s případnými změnami tříd. Automatizovaný proces se pak postará jak o přístup k datům, tak i k jejich změně.

Tento přístup je možné použít i u hotové databáze, kdy opět popíšeme nejdříve kód a mapování.

Jak bylo zmíněno výše, s EF je možné využít i opačného postupu[5]. Ten my ale potřebovat nebudeme. Díky EF můžeme dokonce měnit druh databáze, ve které máme data uložena, aniž bychom museli kód přepisovat. Toho bylo využito i u naší aplikace, když testovací data byla uložena v SQL CE databázi a provozní pak v MSSQL. A díky LINQ můžeme nad daty provádět dotazy pomocí silně typované syntaxe.

3.5 ASP.NET

ASP.NET je komponenta .NET Frameworku zaměřená na vývoj webových aplikací. Je následovníkem starší technologie ASP(Active Server Pages). ASP.NET (jako všechny komponenty .NET) na rozdíl od ASP již běží na CLR. Programátoři tak opět mohou využít jakéhokoliv .NET jazyka [15].

Kód je předkompilován a uložen v assembly, místo toho, aby byl interpretován. Požadavky pak mohou být rychleji zpracovány, protože kód nemusí být neustále znovu parsován [8].

Na tuto aplikaci bylo použito rozšíření ASP.NET MVC Framework 4, na který se blíže podíváme v druhé části následující kapitoly 4.2.

3.6 Visual Studio

Náš projekt byl vyvíjen v Microsoft Visual Studiu. V tomto vývojovém prostředí je možno vytvářet mnoho aplikací, ať už konzolových, s grafickým rozhraním, webových či jiných.

Integrovaný editor kódu obsahuje IntelliSense[6] používající i částečné shody. Obsahuje funkce jako zvýraznění kódu, zoom, generování kódu, použití vlastních snipetů kódu, refaktorování a mnoho dalšího.

Součástí je integrovaný debugger, designer formulářů, webu, tříd a databázových schémat. Obsahuje i WYSIWYG[7] editor.

Studio je samozřejmě rozšiřitelné o mnoho doplňků a po doinstalování Visual Studio customization SDK je můžeme vytvářet sami přímo ve Visual Studiu [9].

3.7 NuGet

NuGet[8] je open source projekt původně vyvíjený firmou Microsoft. Poskytuje jednoduchý způsob jak do vlastního projektu zavést komponenty třetích stran. Komponenty jsou uloženy v repozitáři a vývojář si o ně může požádat skrze

[5]Model-First u nově vznikající databáze a Database-First u již existující.

[6]Inteligentní doplňování kódu

[7]What you see is what you get; Způsob práce, kdy to co upravujete v editoru vypadá stejně jako finální produkt.

[8]http://www.nuget.org/

manager. Ten je dostupný jak v podobě s grafickým rozhraním, tak s příkazovou řádkou.

Komponenty jsou zabaleny do balíčků obsahujících kromě nutných souborů i metadata pro jejich konfiguraci. Instalace je pak velice jednoduchá a rychlá, často si stačí pouze najít kýžený balíček a dát příkaz pro jeho instalaci do projektu.

Manager se stará i o závislosti balíčků na jiných. I aktualizace či odinstalování balíčku je pak poměrně rychlé. Informace o použitých balíčcích je uložena v souboru `Packages.Config` v projektu.

3.8 Git a Bitbucket

Přestože to není s koncovým produktem pevně spojen, je Git[9] natolik užitečným nástrojem, že si zaslouží zmínku. Git je program pro distribuovanou správu verzí. Na rozdíl od jiných systémů, je u něj velice jednoduché pracovat nelineárně tedy vytvářet různé větve. Po pochopení principů je práce s ním velice jednoduchá a program sám je velice rychlý a dostupný pro mnoho platforem. Pro podrobnější informace, včetně toho jak se ovládá, doporučujeme knihu Pro Git [1].

Repozitář s prací samotnou je uložen na BitBucket[10]. Tato služba neposkytuje jen místo pro uložení repozitáře, ale i další služby pro týmovou správu kódů, jako vytváření dokumentace, issues a mnoho dalšího.

3.9 Front-end

3.9.1 jQuery

JQuery[11] je malá, rychlá a na funkcionalitu bohatá JavaScriptová knihovna. Zjednodušuje výběr a manipulaci DOM[12] objektů, manipulaci s kaskádovými styly CSS[13] i použití AJAX[14]. Poskytuje práci s událostmi, efekty a animacemi. Je multiplatformní a podporuje mnoho prohlížečů. Díky jednoduchému API umožňuje programátorovi soustředit se na to, co chce provést, místo aby bojoval s jazykem samotným [16]. I díky snadné rozšiřitelnosti se stala značně používanou. Najdeme ji například i ve Visual Studiu v ASP.NET MVC Frameworku [12].

3.9.2 Bootstrap

Twitter Bootstrap[15] je frond-end framework usnadňující vývoj dynamických webových stránek a aplikací. Obsahuje HTML a CSS šablony pro formuláře, tlačítka, navigace, panely a mnoho dalších prvků uživatelského rozhraní, rovněž i volitelné JavaScriptové rozšíření.

[9] http://git-scm.com/

[10] http://bitbucket.org/

[11] http://jquery.com/

[12] Document Object Model, objektově orientovaná reprezentace HTML dokumentu

[13] Cascading Style Sheets, pro popis způsobu zobrazení HTML elementů

[14] Asynchronous JavaScript and XML, technologie pro vývoj interaktivních webových aplikací bez nutnosti načítat celý obsah

[15] http://getbootstrap.com/

V současnosti je především zaměřen na mobile-first responzivní design, což je způsob vytváření stránek tak, aby se jejich rozložení co nejlépe přizpůsobilo různým zařízením a velikostem obrazovek. Používá dostatečných velikostí pro pohodlné čtení i ovládání pomocí prsty. Přičemž vývoj vychází především z potřeb mobilních zařízení.

3.9.3 Ostatní použité technologie

Pro vykreslování grafů statistik byl použit jQuery plugin JqPlot[16]. Obsahuje funkce jako nastavitelné typů grafu, automatický výpočet trendové linie, zvýrazňování částí, tooltipy a další. Silnou stránkou je i snadná rozšiřitelnost. Abychom docílili jednotného vzhledu checkboxů na všech platformách a dostatečné velikosti pro ovládání prsty, použili jsme jQuery plugin ICheck[17].

[16] http://www.jqplot.com/
[17] http://fronteed.com/iCheck/

4. Model-View-Controller

V této kapitole se blíže podíváme, jak je aplikace postavena. V první části si představíme architekturu, kterou jsme použili, a v druhé jak ji naše aplikace konkrétně implementuje.

Vzájemná reakce softwaru, ať už s uživatelem nebo jiným softwarem, je důležitý aspekt. Mnohdy však s postupem času zjišťujeme, že nám dané rozhraní přestává stačit či že by potřebovalo upravit. Většina uživatelů by asi nechtěla ovládat takovou kalkulačku z příkazové řádky, když jsou již dávno zvyklí na grafické rozhraní, kde si vše pohodlně naklikají. Přitom kód pro samotné výpočty by měl zůstat nezměněn. Sečtení dvou čísel by mělo dát stejný výsledek nehledě na to, jak vypadá rozhraní, skrze které jsme mu příkaz ke sčítání zadali.

Jedním ze způsobů jak se na tyto změny připravit a oddělit uživatelské rozhraní od logiky, je právě Model-View-Controller (MVC).

4.1 Architektura Model-View-Controller

Dříve než si tuto architekturu představíme, je třeba upozornit na jeden problém. MVC není úplně přesně definováno, což může přinést různé nepříjemnosti. Základní myšlenka je sice víceméně stejná, konkrétní provedení se už však často liší. Uvést tak pouze, že jsme použili architekturu MVC, není úplně dostatečné. Naše použití si proto upřesníme v druhé části této kapitoly 4.2.

MVC se snaží aplikaci rozdělit na tři logické části tak, aby na sobě byly co možná nejméně závislé. Jsou jimi Model, View a Controller.

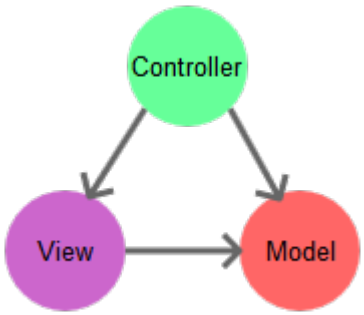
I když se jedna část změní, vliv na ostatní by měl být jen minimální. U našeho příkladu s kalkulačkou nám to umožní přejít od příkazové řádky ke grafickému rozhraní, aniž bychom museli sáhnout do kódu pro výpočet, protože jednotlivé části spolu nejsou pevně svázány.

Oddělení business logiky od uživatelského rozhraní také vede k jednodušší údržbě kódu a jednodušeji se i testuje.

4.1.1 Model

Model je část naší aplikace, která se pravděpodobně bude měnit nejméně. V objektově orientovaném světě to jsou třídy, které modelují problém jež popisujeme. Tak dlouho jak náš problém zůstává stejný, proto není třeba měnit. Naopak uživatelské rozhraní se může měnit docela často, podle toho co je zrovna moderní, používané, doporučené,... Model jsou tedy data a business logika. Často bývá označen jako doménový model. Co je důležité, je to, že model neví o okolním světě nic.

Příkladem mohou být třídy prodavač, klient, produkt,... pokud by se naše aplikace točila okolo prodeje. Do business logiky pak patří, že prodavač předá klientovy zboží až po zaplacení. Pro nás konkrétně to budou třídy reprezentující hráče, ligová kola a pod. O nich se budeme bavit v kapitole 5.



Obrázek 4.1: Základní komunikace mezi modelem, view a controllerem

4.1.2 View

View nám poskytuje pohled na model. Můžeme na něj nahlížet jako na výstup. Příkladem jsou vizuální zobrazení příkazového řádku, grafické rozhraní ale také se na to můžeme dívat jako na pohled pro začátečníka či experta, jak možná znáte z různých antivirových programů.

I třeba takový graf můžeme považovat za view. Jistě si dovedete představit koláčový a výsečový graf reprezentující stejná data. Na tomto příkladu je krásně vidět, že různých views zobrazujících stejná data může být více.

View již o existenci modelu musí vědět a rozumět alespoň něčemu z jeho podstaty. Jinak by těžko bylo schopno prezentovat jeho data [2].

4.1.3 Controller

Controller se stará o uživatelský vstup, manipuluje s view a upravuje data modelu. Je tedy jakousi centrální jednotkou starající se o provázanost.

Příkladem buď upravení čísel pro vytvoření grafu. Controller má na starost, aby se upravili data modelu, vypočetly se nové hodnoty a ty se zobrazili v upraveném grafu.

Controller tak ví o view i modelu, ale model a view o něm nevědí [2].

Obrázek 4.1 zachycuje základní komunikaci mezi jednotlivými částmi.

4.1.4 Variace MVC

Aplikační model je objekt, který ví o existenci views i o jejich potřebě být informován o změnách. Liší se tak od doménového modelu, který o okolním světě neví. Neví sice, kdo všechno chce být informován, ale má mechanizmus, jak je na změnu upozornit [6]. Způsob kterým komunikuje s doménovým modelem se různí. V některých implementacích mají i views vazbu na controller.

4.2 ASP.NET MVC

Námi použitá platforma pro vývoj aplikace je ASP.NET MVC, framework pro vývoj webových aplikací postavených na ASP.NET s užitím návrhového vzoru MVC. Na rozdíl od WebForms, jednou z alternativních platforem pro vývoj

Obrázek 4.2: Fáze zpracováni požadavku v ASP.NET MVC [13]

webových aplikací v ASP.NET, se již nesnaží zavádět stav do bezstavového prostředí. Kromě výhod spojených s MVC se tak i zbavil velkého problému WebForms – značného množství přenášených dat vynucených pro zachování stavu [13].

ASP.NET MVC poskytuje větší kontrolu nad výsledným HTML, než umožňují WebForms [13]. Rendrují přesně to, co je jim dáno. Na jednom view můžeme navíc použít libovolné množství formulářů, což ve WebForms nešlo [13].

4.2.1 Controller v ASP.NET

Standardně se při popisu MVC začíná přiblížením modelu. Jelikož se nám při jeho vysvětlování hodili znalosti o controlleru, řekněme si proto nejprve něco o něm.

Jedním z klíčových aspektů ASP.NET MVC je, že již neexistuje přímý vztah mezi uživatelskými požadavky a fyzickými soubory na webovém serveru. V tradičním pojetí byla žádost o stránku přeložena na zavolání souboru. Při přístupu na adresu „http://mujweb.cz/mastranka.aspx“ byl v kořeni vyhledán soubor mastranka.aspx, zpracován a vrácen HTML kód. To se již neděje.

Když přijde požadavek, například http://mujweb.cz/produkt/detail/1, tak ho komponenta nazývaná routing engine přiřadí k vybrané cestě. Cesta definuje požadavek pomocí svého názvu a stanovuje controller a jeho metodu, která by ho měla zpracovat. Jakmile je cesta identifikována, routing engine vytvoří instanci controller třídy. Ta pak požadavek zpracuje pomocí určené metody. Po zpracování metoda uživateli pošle výsledek, což je typicky HTML vytvořené pomocí view. Tento proces je znázorněn na obrázku 4.2.

Controllrům můžeme nastavit různé atributy. S jejich pomocí tak můžeme snadno aplikovat určitou logiku. Zabraňuje to zbytečnému opakování kódu, který bychom jinam museli psát neustále znovu. My tak například pomocí atributu `Authorize` zakážeme přístup anonymním uživatelům tam, kam je nechceme pustit a pomocí `AllowAnonymous` jim naopak povolíme přístup k metodám, které by jinak byly zakázány. Takovéto filtry je možno aplikovat i globálně.

Routing systém ve skutečnosti nic o controllerech ani jejich metodách neví. Pouze přiřadí požadavek dle zadaného vzorce a pošle tu informaci dál. Tento princip dosáhl takového úspěchu, že byl zpřístupněn i ostatním částem ASP.NET frameworku [13]. Cesta URL v prohlížeči pak může odrážet logiku problému, místo logiky souborového systému.

Controller tak sbírá uživatelské požadavky, zpracovává je a nakonec i vybere které view bude použito pro zobrazení. Může při tom pracovat i s modelem, přidat, změnit či odstranit nějaká data, spustit validaci uživatelských vstupů a podobně.

Controller třídy dědí od základní třídy `Controller` a dle konvence ho mají uvedeno i v názvu na konci. V URL se však slovo „Controller“ neuvádí. „Statis-

tics/Player“ tedy odkazuje na metodu `Player` třídy `StatisticsController`.

4.2.2 Model v ASP.NET

Model v ASP.NET MVC je trojího typu [13].

Datový model

Objekty v datovém modelu reprezentují třídy pro interakci s databází. Tyto třídy mohou být vygenerovány dle tabulek nějaké databáze (Database-first přístup) nebo si je můžeme napsat ručně. My jsme využili ruční přístup, navíc s tím, že tabulky databáze jsme si nechali vygenerovat na základě těchto tříd (Code-first přístup). Jednotlivé části označíme pomocí datových anotací a řekneme tak, jakého typu má daná položka být, zda je povinná a podobně. S pomocí těchto informací je Entity Framework schopen vytvářet tabulky v databázi, ale můžeme je použít i pro validaci či jiné účely.

V našem projektu jsou to například třídy `User` reprezentující uživatele či `LeagueRound` pro ligové kolo.

Business model

Třídy business modelu implementují funkcionalitu reprezentující pravidla a zpracovávají business logiku. Mohou při tom komunikovat s třídami datového modelu kvůli načítání či ukládáni dat.

V našem projektu je to například třída `LeagueRoundDrawing` starající se o rozlosování hráčů do kol dle zadaných pravidel. Potřebuje očividně pracovat s datovým modelem, mimo jiné proto aby byla schopna načíst výsledky a uložit přiřazení hráčů do skupin.

View model

View model shromažďuje data posílaná od controlleru k view. Samotné třídy by neměli obsahovat žádnou vlastní logiku, pouze informace na předání. Sami žádné data nezpracovávají. Jsou také použity pro předání dat opačným směrem, od view ke controlleru, při odesílání formulářů z vyrenderovaného view.

Příkladem takové naší třídy je `LeagueRankModel` použitý pro předání informací nutných k zobrazení žebříčku hráčů.

4.2.3 View v ASP.NET

View soubory jsou použity jako nejčastější možnost pro renderování obsahu k odeslání do uživatelova browseru. Kdybychom chtěli, mohli bychom do nich psát jen HTML kód, nám to ale stačit nebude. Byly bychom tak schopni zobrazovat pouze statický obsah místo dynamického. Proto se do nich přidává server-side kód, který je zpracován a uživateli je pak odeslán výsledek. Existuje více syntaxí pro vkládání server-side kódu do view souborů.

Zpracování má na starosti view engine. Různé engines rozumějí rozdílným formátům souborů. V aplikaci jich můžeme používat více. Standardně jsou v projektu dodány `WebFormsViewEngine` a `RazorViewEngine`. Můžeme si ale doinstalovat mnohé další, stojíme-li o to.

Engine z WebForms je původní engine ASP.NET frameworku pracující se soubory .aspx a .ascx. My budeme používat novější Razor engine, který pracuje s .cshtml pro server-side kód psaný v jazyce C# a .vbhtml v jazyce VB.NET. V našem projektu ovšem používáme pouze jazyk C#.

V razor syntaxi je server-side kód vkládán přímo mezi HTML značky místo do samostatných souborů, na které by se odkazovalo. Nejsou v něm ani žádné serverové ovládací prvky, jako byly ve WebForms (například `<asp:Button />`). Máme tak větší kontrolu nad výsledným HTML. Využívá také méně znaků a je jednodušší na použití [13].

Views mohou být silně typovány view modelem. Máme tak k dispozici IntelliSense, kontrolu při generování i kompilaci kódu. Je možné vytvářet částečné view a ty pak používat opakovaně. I ty mohou být silně typované.

Pro generování HTML můžeme využít i HTML Helper metod. Jejich aplikace je v razor velice snadná, ve view zavoláme metodu a v jejím místě dojde ke vložení výsledného HTML. Přičemž tělo metody je jako ostatní části zkompilováno. Pomocí parametrů jde ovlivňovat výsledek. Příkladem buď zavolání metody `Html.ActionLink`. Jako parametr mu dáme text, jež se má zobrazit, controller a jeho metodu na kterou odkazujeme. Výsledkem bude odpovídající URL.

Je možné si napsat i vlastní helper metody, čehož jsme také využili například pro renderování ligové skupiny. Navíc na rozdíl od partial view, může byt zkompilován jako knihovna a použit ve vícero projektech.

Také je nutno uvést, že tu dochází k porušení klasického MVC. Jistá business logika je totiž aplikována i ve view. Má to ale svůj dobrý důvod. Konkrétně jde o ošetřování vstupů uživatele zadávané do formulářů ve view. Zda je hodnota validní by měl ošetřovat model a ne view samotné, přesto tak ale činí skrze jQuery validaci. Důvodem je zkvalitnění uživatelského rozhraní a zlehčení práce s ním i zamezení zbytečnému toku dat. Pokud by se mělo vše ověřovat až v modelu, je třeba formulář odeslat jako požadavek. Ten je pak na serveru zpracován controllerem, který nechá vstup zkontrolovat modelem. Je-li v něm chyba, je to oznámeno přidanou informací ve view. Uživatel musí chybu opravit a znovu odeslat požadavek. Je-li na pomalém připojení k internetu nebo musí-li za data platit, tak z toho jistě nebude nadšený.

Proto dochází k ošetření vstupů pomocí skriptů už ve view na straně klienta. To značně zpříjemní uživatelský prožitek a omezí zbytečný tok dat.

Porušuje to i to, že by view model neměl nést žádnou logiku. Jelikož je to on, kdo od controlleru předává data do view, tak je to také on, komu se informace o logice musí dodat, aby pak mohla být předána view a vykonána JavaScriptem na straně klienta.

Ani v jednom případě se však nejedná o zásadní provinění. Informace pro validaci jsou třídám nastaveny pomocí atributů a o zbytek se již postará automatizovaný proces. Reagovat na změny jiných částí proto nebývá nijak zvlášť složité ani pracné. Výše zmíněné výhody nám za toto malé provinění stojí.

V žádném případě to ale neznamená, že by nám to nahradilo validaci na straně serveru. To by byla vážná bezpečnostní trhlina, protože jak je známo, nikdy nemáme věřit uživatelskému vstupu [4]. Navíc ne každý browser podporuje či má povolen JavaScript. Některá kontrola ani u uživatele proběhnout nemůže, protože třeba potřebuje data z modelu, která nemůžeme uživateli poskytnout,

aniž bychom se tím vystavili nechtěnému úniku informací.

4.2.4 Struktura projektu

Struktura ASP.NET MVC 4 projektu se řídí jistými konvencemi. Jednotlivé části jsou popsány v tabulce 4.1. Některé složky jsme si dodali sami nad rámec toho, co vzniká automaticky při vytváření projektu.

App_Data	Do této složky patří soubory, u kterých potřebujeme práva ke čtení a zápisu. Může tam být třeba soubor s databází.
App_Start	Tato složka obsahuje konfigurační soubory různých technologií použitých v projektu jako je autentizace, routing, filtering, bundling a pod.
ConfigurationSections	Zde jsou zdrojové kódy našich vlastních rozšíření konfiguračních souborů ASP.NET.
Content	Složka určená pro CSS soubory a podobné položky určené k designu aplikace.
Controllers	Jak název napovídá, jde o složku určenou pro soubory se zdrojovým kódem controllerů.
Filters	Složka určená pro soubory se zdrojovým kódem atributů.
Helpers	Zde jsou soubory se zdrojovým kódem našich vlastních Helperů.
Images	Místo určené pro obrázky.
Localizations	Složka se zdroji lokalizovaných řetězců uživatelského rozhraní.
Models	Tato složka je určena pro zdrojové kódy modelu aplikace.
Services	Zdrojové kódy pro poskytování služeb jako třeba rozesílání emailů.
Scripts	Místo pro JavaScriptové soubory.
Views	Do této složky patří soubory pro uživatelské rozhraní. Kód souborů může být vytvořen ve více syntaxích a zobrazen pomocí více view engines. Soubory pro view patřící k nějakému controlleru jsou umístěny v podsložce nesoucí jeho název bez koncového „Controller“. Společné soubory jsou umístěny ve podsložce Shared. Views soubory jsou použity pro vygenerování koncového HTML poslaného prohlížeči uživatele.
Properties	Dvojklikem na tuto položku se nám otevře okno s vlastnostmi projektu. Můžeme tam kupříkladu změnit na jakou verzi .NET je projekt cílen, upravit možnosti sestavení či publikace projektu. Po rozvinutí se nám zobrazí soubor AssemblyInfo.cs, kde jsou uložena metadata assembly projektu.
References	Obsahuje seznam odkazovaných assemblies v našem projektu.
Web.Config	Konfigurační soubor v XML formátu obsahující nastavení ASP.NET aplikace.
Global.asax	Soubor pro deklaraci a obsluhu událostí na úrovni aplikace a session.
Packages.Config	Soubor v XML formátu obsahující informace o nainstalovaných NuGet balíčcích v naší aplikaci.

Tabulka 4.1: Popis částí ASP.NET MVC 4 projektu [13]

5. Návrh aplikace, tříd a algoritmů

Jak jsme si řekli v předchozí kapitole o MVC, model úzce souvisí s problémem který řešíme. Proto si tuto část popíšeme blíž. Popisovaná verze aplikace je v1.0.2.

Verze použitých technologií a nástrojů jsou uvedeny v tabulce 5.1. Technologie samotné byly představeny v kapitole 3.

5.1 Ligový model

Přibližme si nejdříve model ligy, protože to bude jedna se základních věcí, se kterou tu budeme pracovat.

Ligou myslíme nějakou dlouhodobou soutěž jednotlivců, například amatérskou badmintonovou ligu či squashovou ligu juniorů, probíhající v nějakém sportovním zařízení. Každá taková liga si nese svůj název. Dlouhodobou myslíme to, že probíhá více let. Samozřejmě, aby bylo o co soutěžit, je jednou za čas ukončena, vyhodnocena a znovu započata. Rozděluje se tedy na *ročníky*. Typické bývá, že jeden ročník trvá o něco méně než rok, aby byl prostor pro mimo sezónní pauzu.

Stejně jako liga i ročník si nese svůj název. Jedním z důvodů může být pojmenování ročníku podle sponzora, pokud takového centrum má.

Jeden ročník je složen z *kol*. V rámci každého kola jsou hráči rozděleni do *skupin* podle své výkonnosti, kde se hraje systémem každý s každým. Nejlepší hráči jsou ve skupině první, dále pak výkonnost klesá.

V našem modelu jím odpovídají třídy `League` pro ligu, `LeagueSeason` pro ročník, `LeagueRound` pro kolo a `LeagueGroup` pro skupinu. Jejich hierarchie je vidět na obrázcích 5.1 a 5.3.

Co je také poměrně běžné je pořádání *turnajů* v rámci ligy. Ty ale nemusejí být nutně otevřené jen pro hráče ligy. Ne každý má čas účastnit se ligy, ale mnozí mají zájem jít si zahrát na turnaj. Je to navíc krásná příležitost ukázat ligu veřejnosti. Běžný hráč se tak potká s ligovými, okusí jejich úroveň, která mu třeba zachutná natolik, že se sám nakonec přidá. Výsledky takových hráčů se pak logicky do ligy nepočítají. Opět takové turnaje bývají pojmenované.

Centrum, zcela přirozeně, stojí o nábor dalších hráčů, je ale třeba zahrnout i jak reálný svět funguje. Lidi se stěhují, zraňují a mění se jim časové možnosti až

Náš projekt	v1.0.2
.NET Framework	4.0
Entity Framework	5.0.0
Visual Studio	2010 SP1
jQuery	v2.1.1
Bootstrap	v3.3.1
JqPlot	v1.0.8
iCheck	v1.0.2

Tabulka 5.1: Přehled verzí technologií a nástrojů použitých v našem projektu

Obrázek 5.1: Systém ligových kol, počet kol i turnajů může být libovolně velký. V lize právě probíhá druhé kolo, třetí je připravené, ale není ještě rozlosované.

Obrázek 5.2: Životní fáze kola

třeba tak, že nejsou slučitelné s hraním ligy. Hráči nám mohou přibývat i ubývat.

Shrňme si co z toho vyplývá pro náš model a potažmo aplikaci.

- Aplikace by měla zvládat více než jednu ligu.
- V rámci ligy se mohou libovolně přidávat ročníky.
- Ročník může obsahovat libovolné množství kol a turnajů.
- V jednom kole může být libovolný počet skupin a velikost těchto skupin může být různě velká.
- Výsledky jsou potřeba mezi koly vyhodnotit, zahrnout odchody a příchody hráčů a připravit kolo nové.
- Turnaje se mohou překrývat s koly.
- Kola v rámci jednoho ročníku se naopak překrývat nemohou.

To vše bylo třeba vzít v úvahu při vytváření modelu. Přičemž se bude třeba vypořádat s těžkými problémy vyplývajícími z různých požadavků rozdílných center a sportů.

Centrem dění hráčů jsou především kola, skupiny a turnaje. Pojďme se na ně proto podívat ještě trochu blíže.

5.1.1 Kola

Kolo může existovat ve třech různých životních fázích. Nejprve dojde k jeho vytvoření administrátorem ve stavu nerozlosované. Takové kolo ještě neobsahuje žádné skupiny. Můžeme si takto vytvořit více za sebou jdoucích kol.

Do dalšího stavu se dostane po akci nazývané *rozlosování* a stane se tak aktuální[1] (neboli rozlosované, probíhající). Je to kolo, které je právě hrané. Samozřejmě již obsahuje jak skupiny, tak hráče v nich. Každé kolo má určený datum, kdy začíná a datum, kdy končí. Hráči mohou zadávat výsledky zápasů právě v tomto termínu. Toto časové omezení ale neurčuje stav ve kterém se nachází. Kolo může být aktuální, přestože čas začátku ještě nenastal. Nebo naopak mohlo datumově už skončit, ale stále být aktuální. Tyto přesahy lize typicky slouží jako doba pro kontrolu výsledků a případné reklamace. Již ale nechceme, aby hráči mohli výsledky libovolně měnit. Administrátor tak ale učinit může.

Posledním stavem je ukončené. Do této fáze se kolo dostane při rozlosování kola následujícího či ukončení ročníku. A v tomto stavu již setrvá[2]. Pouze výsledky uzavřených kol se počítají do žebříku hráčů.

Termín kola je viditelný ve všech životních fázích kola. V nerozlosovaném stavu tato informace slouží jako harmonogram průběhu ligy, v aktuálním jako doba, po kterou hráči mohou zadávat výsledky a v ukončeném jako informace pro historii ligy.

Přibližme si ještě rozlosování. Je to proces, při kterém se aktuální kolo ukončí a následující nerozlosované se stane aktuálním. Dojde k vyhodnocení probíhajícího kola, tzn. určí se pořadí hráčů ve skupinách. Nejlepší hráči postoupí do dalšího kola do vyšších skupin, nejhorší spadnou níže a ostatní zůstanou. Vytvoří se skupiny následujícího kola, přiřadí se do nich hráči dle postupů, současné kolo se ukončí a z nerozlosovaného kola se stává aktuální. Při tomto procesu se také provedou příchody a odchody hráčů.

Výjimkou je první kolo ročníku. To žádné předchozí kola na vyhodnocení nemá, takže je situace jednodušší. Prostě se vytvoří skupiny, přidají se do nich hráči a kolo se stane aktuálním.

Aktuální kolo může být v ročníku v danou chvíli pouze jedno, ale probíhat může více lig či ročníků současně. Jelikož jsou aktuální kola jedním ze základních center dění, mají v aplikaci svůj vlastní přehled.

5.1.2 Skupiny

Jak bylo uvedeno, hráči jsou v každém kole rozděleni do skupin. Ty mají nějakou standardní velikost. Jeden hráč může být v jednom kole pouze v jedné skupině, může ale samozřejmě hrát ve více ligách současně. Body se udělují za zápasy i za skupinu, ve které je hráč umístěn. Čím vyšší skupina, tím bývá bodů více, ale to je již v kompetenci administrátora, jaký počet bodů které udělí.

Navíc je možné v každé skupině udělovat různý počet bodů za tentýž výsledek. Více je uvedeno v sekci 5.2.

Hráč může být ve skupině umístěn dvěma způsoby - jako normální hráč či jako hráč s udělenou *divokou kartou*. Divoká karta je způsob, jak umožnit hráčům rychlejší postup v lize a zachovat přitom jistou férovost. Běžným způsobem přidání nových hráčů, je jejich umístění do poslední skupiny. Co ale dělat, když ale jeho úroveň převyšuje ostatní? Při větším počtů skupin, v což centrum pravděpodobně doufá, by vystoupání na dostatečnou úroveň mohlo trvat značnou dobu. Obzvláště

[1] V době ručních úprav rozlosování může kolo být ještě nerozlosované ale už aktuální, viz sekce 5.4

[2] Výjimkou je funkce umožňující návrat k předchozímu kolu, viz sekce 5.5

Obrázek 5.3: Skupiny v rámci kola, kde hraje se systémem na tři vítězné sety.

pokud se postupuje pouze o jednu skupinu. Jenže přidání rovnou do vyšší skupiny by nemuselo být fér vůči ostatním, kteří si svou pozici vybojovali. Řešením je udělení divoké karty do vybrané skupiny. Daný sportovec je umístěn jako osoba navíc, velikost skupiny je tedy o jednoho člověka větší, než je standardní. Svým výkonem musí přesvědčit, že do skupiny patří. Třeba tak, že vyhraje všechny zápasy. Pokud se mu to podaří splnit, kartu obhájil a ve skupině v příštím kole zůstane jako normální hráč. Když se tak nestane, padá do poslední.

Uvedený způsob je sice běžný, nicméně vynuceno to není. Přidávat můžeme i normálně přímo. Hráč pak není umístěn nad limit a rovnou postupuje jako ostatní.

Jak bylo uvedeno, skupinu reprezentuje třída `LeagueGroup`, o přiřazení hráče do skupiny se stará třída `UserInGroup`.

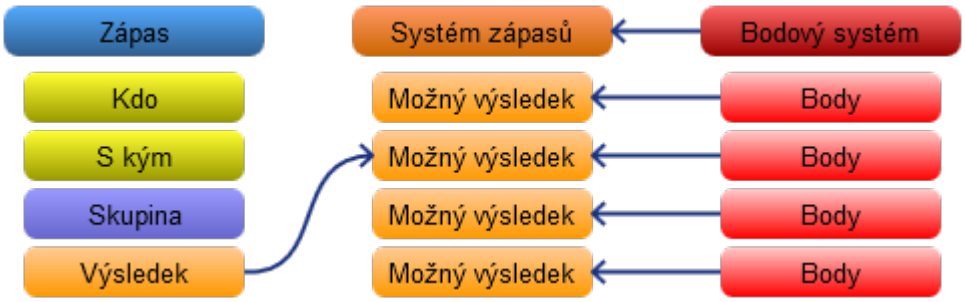
5.1.3 Turnaje

Autor z nabraných zkušeností došel, mimo jiné, k jednomu poznatku ohledně turnajů. Je opravdu třeba rozlišovat turnaje lokální úrovně od těch ostatních oficiálnějších. Přes organizaci po průběh i atmosféru.

Zatímco žádného účastníka turnaje Áčkové úrovně neuvidíte popíjet pivo, při turnajích lokálních je to běžné, někdy se i očekává, že takové občerstvení bude zajištěno. To celkem příhodně odráží atmosféru. Hráči se na takový turnaj jdou především bavit. Pro centrum i ligu to je příležitost, jak přibrat další zákazníky. Přístup je proto mnohem volnější, jde o to si zahrát a pobavit se.

Systém hraní není výjimkou. Často vzniká na poslední chvíli. I počet účastníků se často mění ještě v den turnaje. Přece jen se nejedná o mistrovství republiky, tak si můžeme dovolit být trochu lidští. Přece jen jde především o to dobře si zahrát. Změna počtu je ale někdy tak značná, že to předem zvolený systém položí na kolena.

Nejde tu o žádné sportem se živící profesionály, ale o normální lidi, kteří nemohou veškerý čas věnovat sportu. Někdo nemůže hrát až do konce, jiný se zraní a raději odstoupí. K úpravám detailů, tak dochází kolikrát i za běhu, aby nenastala situace, kdy je většina zápasů rozhodnuta tím, že někdo odstoupil. To opět může čtenáři připadat podivné, pokud to nikdy nezažil. Velké provinění to ale není. Hráči si často sami odhlasují, zda o nějakou změnu stojí. A o ně tu jde především.



Obrázek 5.4: Jak na sebe navazují zápasy, systémy zápasů a bodové systémy

Organizace tak velice často probíhá s tužkou a papírem. A mnohdy to je mnohem pohodlnější, než mít na to nějaký software.

Naše aplikace má organizaci ulehčit a ne vnucovat něco, o co nikdo nestojí. Z těchto důvodů jsme se rozhodli podporovat turnaje jen na úrovni organizace doby konání a zadávání výsledků. Ať si centrum samo zvolí svůj oblíbený systém či tak častou tužku a papír. Jelikož jde o krátkodobou akci, je to často příhodnější.

Do aplikace proto zadá administrátor informace o turnaji, jako i datum začátku a konce. Data opět budou vidět v přehledu a mohou opět sloužit i jako harmonogram.

Po odehrání se už jen zadají výsledné pozice (či intervaly pozic) a získané body. Turnaj má tak jen 2 fáze, zadaný a odehraný. Přičemž do žebříčku se počítají logicky jen odehrané. Na rozdíl od kol, turnaje se mohou účastnit i hráči, jež ligu nehrají. I s takovýma je počítáno. Zadávají se jako anonymní hráči a body se jim nepočítají.

V jednom ročníku může být libovolný počet turnajů. Znamená to tedy, že se nemusí konat žádný i to, že může být ročník složen jen ze samých turnajů bez jediného kola. Centrum tak může pořádat sérii turnajů, kde se vítěz určí na základě nejvíce nasbíraných bodů.

Turnaj nám reprezentuje třída `Tournament` a výslednou pozici jednoho hráče v něm pak `TournamentResult`. Aby se vyhovělo výstupům různým systémům hraní, tak hráči mohou výslednou pozici i sdílet.

5.2 Zápasy, systémy zápasů a bodové systémy

V této části se podíváme, jak jsme se vypořádali s problémem rozdílných nároků sportů na utkání.

První, co je třeba rozhodnou, je to, do jaké hloubky budeme zápasy evidovat. Stačí nám pouze výsledek setů, nebo budeme ukládat i body každého z nich. Nesmíme u toho však zapomenout, komu má aplikace sloužit. Podívali jsme se proto do praxe a rychle byly překvapeni, že odpověď je nasnadě. Stačí se pohybovat na úrovni setů, body není třeba uchovávat.

Chceme-li zjistit proč, zjistíme, že odpovědí je opět pohodlí. Pokud jste sami nějaký takový zápas hráli, tak jste to dost možná zažily. Dohrajete a už sami nevíte, kolik bodů kdo uhrál. Autor dělal, pravda ne vždy úmyslně, pokusy na toto téma. Zadáme hráčům na turnaji povinnost zapisovat i body. Dokonce ke každému kurtu na to dáme tabulku, do které je mohou rovnou zapsat. Výsledkem je, že si

moc nepomůžeme. Stejně to buď vyplněné není, nebo slyšet, jak si hráči marně snaží vzpomenout, dohadují se či si rovnou vymýšlí stavy, jak to plus mínus mohlo být. Evidovat to proto nemá smysl, uživatelům je to nepohodlné, často to neodráží skutečný stav a ligy to tak prostě nedělají. Máme vycházet z reálných potřeb, tak jsme to odrazili v návrhu.

Přesto jsme si nechali skulinku dobrým návrhem. Část kódu zodpovědná za rozhraní pro zadávání výsledků je separátní, stejně jako jejich vyhodnocování. Kdyby někdy nastala potřeba rozšíření, bude to možné.

5.2.1 Systémy zápasů

Další těžkou otázkou je, jak si poradit s rozdílnými nároky různých sportů. To řešíme nastavitelnými systémy zápasů a jejich oddělením od modelu bodů. Ani zápas není přímo svázán s výsledkem, pouze na něj odkazuje.

Některý sport se hraje na dva vítězné, jiný na tři, čtyři, někdy je zase třeba z časových důvodů hrát jen na dva hrané sety. Někdo označuje výhru tři dva jako 3:2 jiný jako 3x2 a podobně. Administrátor proto může definovat libovolný počet vlastních systémů zápasů. Vytvoří prostě nový systém, pojmenuje si ho a sám nadefinuje možnosti, jak v něm může zápas dopadnout. Opět může dodat libovolný počet možností. U nich nadefinuje kolik setů hráč vyhrál, zda to označuje výsledek zápasů, který byl odehraný či zda šlo o skreč (zápas který se neodehrál a došlo tak ke kontumační výhře). Ke každé možnosti si sám zvolí text, jež má být zobrazen.

5.2.2 Bodové systémy

Jak jsme uvedli před malou chvílí, došlo k oddělení bodů od samotných výsledků. Důvody jsou k tomu následující. Né vždy potřebujeme vědět, kolik bodů je za daný výsledek. Zadávači výsledků je to jedno, ten potřebuje znát pouze výsledek, co má zadat. Jeden systém zápasů může mít více bodových systémů. Jedna liga či ročník může chtít dát výsledku 0:2 jeden bod a jiná třeba žádný. Oba ale shodně používají systém na dva vítězné. Když jde o tentýž systém, tak proč ho mít uložený duplicitně a plýtvat tak místem. Ztratili bychom i informaci, že ligy byly hrány stejným systémem. Otevírá nám to i prostor pro porovnávání výsledků stejných systémů nehledě na počet udělených bodů za ně. Nebo pro funkcionalitu typu, jak by skupina dopadla, kdyby byly body přiřazeny jinak. A finálně, kdybychom do nich vkládali body rovnou, ztratili bychom informaci, jakým bodovým systémem byl zápas hrán, což může být také velice cenná informace.

5.2.3 Zápasy

Ze stejných důvodů i zápas odkazuje pouze na výsledek jak dopadl, místo aby to bylo uvedeno v něm přímo. Opět tak šetříme místem a zachováváme si informaci jakým systémem se hrálo. Každé skupině můžeme také přiřadit jiný bodový systém pro stejný systém zápasů.

Zápas samotný může být ve čtyřech fázích neznámý, nepotvrzený, potvrzený a konfliktní. Neznámý, jak název napovídá, je zápas jehož výsledek ještě nebyl zadán. Nepotvrzený je sice zadán, ale jen jedním ze soupeřů. Potvrzený byl shodně

zadán oběma soupeři zatímco konfliktní je, pokud každý soupeř zadal výsledek jiný.

Uživatelé mohou v termínu kola měnit či rušit výsledky až do jejich potvrzení. Pokud by náhodou udělali oba stejnou chybu při zadávání, musí už o změnu požádat správce ligy.

5.2.4 Implementace

Systém zápasů reprezentuje třída `MatchSystem`. Jeden možný výsledek v systému představuje `MatchOutcome`. V něm jsou uloženy potřebné informace pro interpretaci - kolik setů hráč získal, zda došlo ke hře, nebo se jedná o skreč, zda jde o výhru, text který se má uživateli zobrazit a odkaz na opačný výsledek. Vše je vedeno z pohledu hráče. Pohled soupeře je dán výsledkem opačným. Takže počet prohraných setů je počet vyhraných setů opačného výsledku.

Na první pohled by se mohlo zdát, že zde nejsou vedeny remízy, protože si pamatujeme pouze informaci, zda to bylo vítězství či prohra. To ale není pravda, právě díky opačnému výsledku. Jsou-li shodně výsledek i opačný výsledek k němu oba vítězstvím či prohrou, jde o remízu.

Aby došlo k odstínění implementace, výsledky se interpretují pomocí třídy `MatchOutcomeEvaluator`. Ta datovému modelu výsledků dodává business logiku. Interpretuje o jaký druh výsledku se jedná, zda jde o výhru a podobně. Pomocí tohoto aplikačního modelu s výsledky pracujeme. Máme tam možnost změnit implementaci, kdybychom někdy přece jen došli k názoru, že by se nám hodilo ukládat výsledky na hlubší úrovni.

Business logiku ukládání a validace zápasů má na starosti třída `MatchSaver`. Tento aplikační model se stará o samotné ukládání, změnu, rušení a potvrzování výsledků. Hlídá, aby nedošlo k mixování různých systémů zápasů nebo aby se uživatel nepokusil odehrát zápas sám se sebou. Logika je rozdílná když výsledek ukládá hráč nebo správce. Hráči například nedovolí uložit zápas do kola mimo termín. Správce tak ale učinit může. Máme tak k dispozici centralizované rozhraní. Na něm můžeme stavět dál, když bychom se rozhodli dále aplikaci rozšiřovat, třeba modulem pro Android.

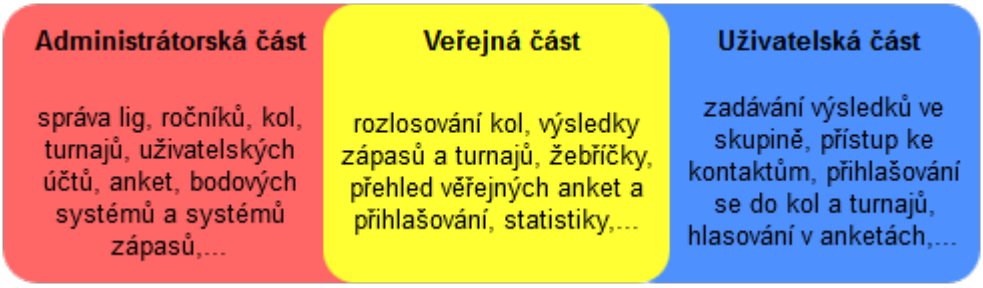
Bodový systém reprezentuje třída `MatchPointsSystem` a body jednotlivým výsledkům přiděluje `MatchOutcomePoints`.

5.3 Rozdělení aplikace

Nyní si představíme, jak je aplikace samotná rozdělená. Čtenář si teď možná pomyslí, proč ne už dříve. Z prostého důvodu, při čtení pojmů jako je správa kol a rozlosování, by nemuselo dojít k úplnému vzájemnému pochopení. Třeba by to vyvolalo představu kola jako v hokeji či fotbale, kde se v rámci jednoho kola odehrávají zápasy dvojic. Což sice není úplně špatně, svým způsobem to odpovídá našim skupinám o velikosti dva. Ale už sem nezapadá myšlenka postupů a pádů mezi každým kolem.

Aplikace rozlišuje uživatele trojího druhu a tomu také odpovídá její rozdělení na tři části:

1. Veřejná část pro anonymní návštěvníky



Obrázek 5.5: Aplikace má tři části. Veřejnou pro anonymní návštěvníky, uživatelskou pro hráče a administrátorskou pro správce.

2. Uživatelská část pro hráče ligy

3. Administrátorská část pro správce ligy

Veřejná část je volně dostupná každému. Patří do ní věci jako přehled aktuálních kol, historie hráčů, kol a turnajů, statistiky, veřejné ankety a zveřejněné přihlašování hráčů, žebříčky hráčů a podobně.

Hráčům ligy je vytvořen normální uživatelský účet, se kterým získají přístup do uživatelské části. Tam mohou provádět akce jako zadávání výsledků v aktuálních kolech, kterých se účastní, hlasování v anketách, přihlašování se do kol a turnajů, kde to bylo umožněno, zjišťovat kontakty na své spoluhráče a tak dále.

Správci mají přístup do administrátorské části. Z bezpečnostních důvodů nemají přístup do části hráčské a nemohou se ani ligy zúčastnit. To samozřejmě neznamená, že by ligu nemohli hrát, jen si musí vytvořit k těmto účelům normální hráčský účet.

Správci mohou provádět nejvíce akcí. Mají na starost správu lig, ročníků kol a turnajů, provádějí rozlosování kol, nastavují pravidla, zajišťují příchody a odchody hráčů v lize, spravují účty a mnoho dalšího. Co vše kdo může dělat a jak, lze najít v samostatné uživatelské dokumentaci.

V kapitole Model-View-Controller, jsme se dozvěděli, jak je náš projekt členěn. V tabulkách 5.2 a 5.3 je uveden přehled těch nejdůležitějších controllerů a tříd modelu.

5.4 Rozlosování

Rozlosování je jednou z klíčových funkcí, která zároveň musí vyřešit obtížný problém splnit různé požadavky rozdílných sportů i pravidel lig. Podívejme se na něj ještě detailněji a ukažme si, jak jsme se s tímto problémem vypořádali.

Rozlosování je trojího druhu automatické, poloautomatické a ruční (neboli manuální). Na starosti ho má třída `LeagueRoundDrawing`.

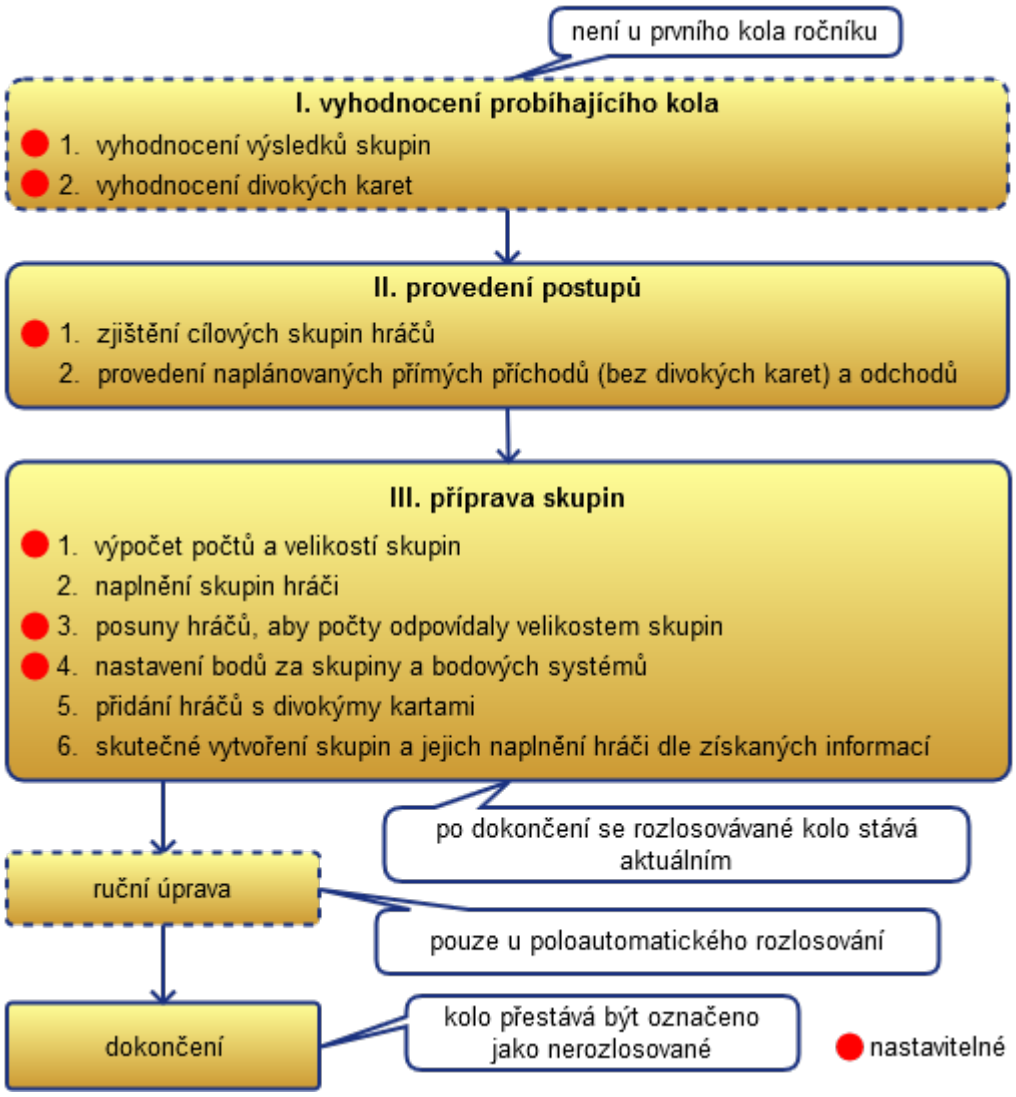
Podívejme se nejprve na to automatické, co musí provést a co vše u toho musí řešit. Grafické znázornění je na obrázku 5.6.

AccountController	Má na starosti akce s uživatelskými účty, změnu uživatelského nastavení a pod.
AdministrationController	Má na starosti vše okolo administrace ligy, vytváření lig, ročníků a kol, správy systémů zápasů a bodových systémů,
EnrollmentController	Umožňuje přihlašování se do kol a turnajů.
EventController	Zobrazení zalogovaných událostí.
LeagueController	Zobrazování ligových kol a skupin, žebříčku hráčů, uživatelské ukládání výsledků a další akce okolo ligy pro běžné uživatele.
StatisticsController	Zobrazování statistik hráčů, výsledků, kol,...
TournamentController	Zobrazování výsledků turnajů.
VotingController	Má na starosti hlasování v anketách.

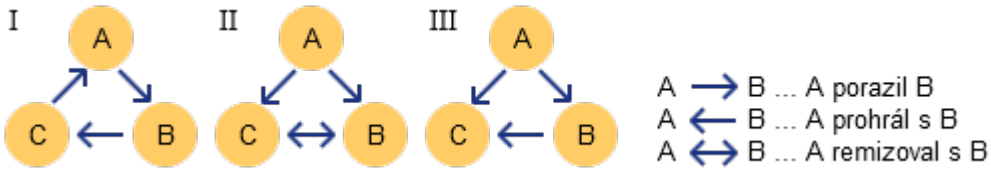
Tabulka 5.2: Nejdůležitější controllery

EventLogger	Model logující události v aplikaci jako je třeba přidání výsledku zápasu, úprava uživatelského účtu a pod.
GroupResultsEvaluator	Model pro vyhodnocování zápasů ve skupině
League	Třída pro model ligy
LeagueGroup	Třída představující ligovou skupinu
LeagueRound	Třída ligového kola
LeagueSeason	Třída ročníku ligy
LRDSettings	Model pro ukládání nastavení rozlosování kol
Match	Třída reprezentující zápas
MatchOutcome	Třída představující možný výsledek zápasu
MatchOutcomeEvaluator	Třída pro vyhodnocení výsledků zápasu, zda se jedná o vítězství, prohru a pod.
MatchOutcomePoints	Třída přiřazující body výsledku zápasu
MatchPointSystem	Třída pro bodový systém výsledků zápasu
MatchSaver	Stará se o ukládání zápasů, ať už jako uživatel či jako administrátor
MatchSystem	Třída reprezentující systém zápasů, například na dva vítězné, na tři hrané a pod., záleží na administrátorovi ligy, jaké všechny vytvoří
Tournament	Reprezentuje turnaj
TournamentResult	Představuje výsledek hráče v turnaji
UnconfirmedMatch	Představuje nepotvrzený výsledek zápasu.
User	Třída reprezentující uživatele systému
UserInGroup	Třída reprezentující přiřazení hráče do nějaké skupiny v ligovém kole
UserSettings	Třída pro uživatelské nastavení
Vote	Reprezentuje hlas v anketě
Voting	Třída reprezentující anketu

Tabulka 5.3: Nejdůležitější třídy datového a aplikačního modelu



Obrázek 5.6: Fáze automatického a poloautomatického rozlosování



Obrázek 5.7: Různé situace vzájemných zápasů třech hráčů. V situaci I pravidlo o vzájemném zápasu rozhodnout nemůže. V situaci II je hráč A lepší než oba zbývající, které ale rozdělit nedokáže. V situaci III jsou hráči seřazeni od nejlepšího v pořadí A, B, C.

5.4.1 Vyhodnocení výsledků probíhajícího kola

Vyhodnocení výsledků skupin

Nejdříve se musí vyhodnotit výsledky aktuálního kola. Tato fáze samozřejmě odpadá, jedná-li se o první kolo ročníku. Skládá se ze dvou částí. Tou první je vyhodnocení výsledků ve skupinách, neboli seřazení hráčů ve skupině, a druhou je vyhodnocení úspěšných obhájení divokých karet. Problémem je, že každá liga či sport může mít nároky jiné, neexistuje jediné řešení. Někdo dává přednost nasbíraným bodům, jiný počtu vítězství. A co provést, když nastane shoda? My jsme proto zavedli sadu pravidel pro vyhodnocení. Každý ročník může mít nastavena jiná pravidla a v různých počtech. Správce ligy definuje jejich počet i pořadí. Nerozhodne-li první, přechází se na další. Může tak například nastavit jako prioritní počet nasbíraných bodů pak počet vítězství, méně skrečovaných zápasů, poměr setů, vzájemný zápas, postavení v žebříčku v minulém kole a podobně.

Algoritmus vyhodnocení vzájemných zápasů

Podrobnější pohled si zaslouží pravidlo vzájemného zápasu. Nejde o prosté porovnání dvou hráčů a rozhodnutí pořadí na základě výhry, prohry či nerozhodnutí když jde o remízu. To by pro tři hráče již nefungovalo, jelikož relace vítězství není tranzitivní ve smyslu: Hráč A porazí hráče B, ten porazí hráče C, který pro změnu porazí hráče A (situace I na obrázku 5.7). Je vidět, že zde došlo k utvoření cyklu a neplatí, že když jeden porazí druhého a ten zas třetího, tak první musí porazit třetího. Teď už toto pravidlo samotné opravdu rozhodnout nemůže, kdo z nich je lepší. Pokud ovšem hráč A porazí hráče B i C, kteří spolu remizujou (situace II na obrázku 5.7), pak by mělo určit, že A je lepší než oba zbývající. Hráče B a C ale nerozliší. A konečně, když hráč A opět porazí oba dva a B porazí hráče C (situace III na obrázku 5.7), pak by mělo určit pořadí hráčů A, B, C.

Podobně i pro více hráčů. Pravidlo vytvoří orientovaný graf, ve kterém jsou hráči vrcholy. Hrana pak vede od hráče vítězného k poraženému nebo v obou směrech jde-li o remízu. V tomto grafu najde *silně souvislé komponenty*, které pak uspořádá. Pro samotné vyhledání je použit algoritmus průchodu do hloubky [17]. Tranzitivita mezi komponentami již platí, protože zápas mezi každýma dvěma hráči mohl dopadnout pouze výhrou, prohrou nebo remízou. Hráče v jedné komponentě nemůže rozlišit a tak se na ně musí aplikovat další pravidlo.

Silně souvislé grafy Řekneme, že orientovaný graf G je *silně souvislý*, jestliže pro každé dva vrcholy u, v existuje orientovaná cesta z vrcholu u do vrcholu v a orientovaná cesta z vrcholu v do vrcholu u [3].

Silné souvislé komponenty Je dán orientovaný graf G. Množina vrcholů K se nazývá *silně souvislá komponenta*, též komponenta silné souvislosti, jestliže podgraf indukovaný K je silně souvislý a přidáním libovolného vrcholu ke K přestane indukovaný podgraf být silně souvislý [3].

Zařazení do tříd

O výpočty výsledků skupiny se stará třída `GroupResultsEvaluator`. Je jí možné nastavit i různé strategie výpočtů jako třeba: počítej pouze potvrzené výsledky,

ber nepotvrzené jako potvrzené a podobně. Takže ji můžeme použít i v průběhu kola, kdy mnoho výsledků ještě není známo. Samotným výsledkům zápasů rozumí třída `MatchOutcomeEvaluator`.

Stojí za zmínku, že jednou z raných možností řešení bylo bráno v úvahu nechat správce napsat nějaký skript na seřazení. To by dalo do ruky uživatelům opravdu mocný, ale i nebezpečný nástroj. Tato funkcionalita ale byla poměrně rychle zavržena. Hodně by se omezil počet uživatelů, co by ji byly schopni ovládat, což jde proti základní motivaci tohoto projektu.

Řešení hraničních situací

Především proto, že si druh pravidel, jejich počet i pořadí nastavuje sám správce, může se stát, že ani po použití všech pravidel stejně nedojde k finálnímu rozdělení hráčů. Přesto je nějak musíme seřadit. Vyzývat k nějaké dohrávce není z praktického hlediska možné. Hráči si v ligách určují časy zápasů sami a ne vždy by se jim dařilo se včas sejít a celá liga by tím byla pozdržena. Musíme se tedy rozhodnout sami. Možností je použít náhody, tak ale v našem případě nečiníme, aby byl zachován determinismus. Pokud tedy takový případ nastane, rozdělí se hráči pomocí identifikačního čísla a přednost tak dostanou ti, jejichž účet vznikl dříve. Toto chování by samozřejmě mohlo být přizpůsobeno dle požadavků konkrétního centra, kde by byla aplikace nasazena.

Každé pravidlo tak bere skupinu hráčů, kteří ještě nejsou rozděleni a rozdělí ji na další setříděné skupiny. Hráči v jedné skupině od sebe ještě nebyli dostupnými pravidly rozlišeni. Na každou takto vzniklou skupinu s alespoň dvěma hráči použijeme další pravidlo v řadě. Skončíme až když jsou všechny skupiny jednoprvkové nebo když došla pravidla. V takovém případě je použito výše popsané finální kritérium.

Vyhodnocení divokých karet

Podobnému problému čelí i při vyhodnocování úspěšnosti obhájení divokých karet a podobně ho také řeší. Stejný či podobný přístup je v aplikaci použit ještě několikrát, což dodává jistou konzistenci. Přístup pomocí pravidel navíc přímo odpovídá tomu, jak je liga provozována a umožňuje tak ke každému ročníku automaticky vypisovat pravidla. Přestože jistě bude mít centrum pravidla svá a mnohem obsáhlejší, řešící detaily i mimo sféru tohoto programu, toto hráčům dodá ty nejzajímavější informace, které se jich budou neustále týkat.

Odpovědí tentokrát není pořadí hráčů, ale zda došlo k obhajobě či ne. Zadáním tedy není pořadí pravidel, ale logický výraz z pravidel. Zadávány jsou v disjunktivně normální formě bez negace. Negace není třeba, protože je obsažena v pravidlech. Příklad:

$$(A \wedge B \wedge C) \vee (A \wedge D)$$

Kde A,B,C,D jsou pravidla.

To se uživatelům i dobře představuje: pro obhajobu musí být splněny všechny pravidla z první skupiny, nebo všechna z druhé, třetí,... Skupinou pravidel myslíme všechna pravidla v jedné závorce. Pravidla jsou tentokrát parametrizována a mohou se vyskytovat ve více skupinách (pravidlo A v příkladu). Správce může kupříkladu zvolit pravidla typu: Vyhrát X procent zápasů, skončit ve skupině alespoň na *Y-tém* místě a podobně.

Algoritmus pro vyhodnocení pak probíhá následovně:

```
for SkupinaPravidel ∈ PravidlaProObhajobuDK do
    res ← Obhájeno;
    for Pravidelo ∈ SkupinaPravidel do
        if neplatí(Pravidlo) then
            res ← Neobhájeno;
        end
    end
    if res == Obhájeno then
        return Obhájeno;
    end
end
return Neobhájeno
```

Algoritmus 1: Vyhodnocení divoké karty

5.4.2 Provedení postupů

Druhou fází rozlosování je uskutečnění postupů, jak je vidět v pokračování na obrázku 5.6. Na základě sebraných informací hráči postupují o skupiny výše, níže, nebo v ní zůstávají. Znovu je možno nastavit, kolik hráčů postupuje a o kolik skupin. A do skupiny se také napevno začlení i ti, co obhájili svou divokou kartu. Ti, co to nezvládli, jsou posláni do skupiny poslední. Pokud se ale postupuje o více než jednu skupinu, nastane situace, kdy by hráč měl postoupit do skupiny, která už neexistuje (třeba do nulté). To se samozřejmě stát nemůže. Jde tedy do první skupiny jako nejvyšší možné, kde vzniká potenciální „přetlak“ hráčů. Totéž se děje s poslední skupinou a padajícími hráči. A v některých skupinách pod první/nad poslední je naopak hráčů méně. Tímto faktem se bude zabývat až další fáze.

Další možné přebytky či nedostatky hráčů přináší druhá část této fáze a tou je uskutečnění naplánovaných příchodů a odchodů hráčů. Hráči jsou zde přibíráni jen přímo bez divokých karet. Hráči s divokou kartou budou přidáni ve fázi následující, protože to jsou hráči nad limit skupiny.

5.4.3 Příprava skupin

Příprava skupin

Třetí fáze začíná výpočtem kolik skupin má vzniknout a jaké mají být jejich velikosti. I tento krok je nastavitelný pro každý ročník zvlášť. Potřebujeme totiž vyřešit co dělat, když hráči nejdou beze zbytku rozdělit do skupin.

Standardní velikost skupiny je jen jedním z nastavitelných parametrů. Můžeme specifikovat, při jakém přebytku se má vytvářet skupina nová. Nastavitelné je i mezi kolik posledních skupin se má přebytek rozdělovat.

Dalším krokem je naplnění hráčů do zatím virtuálních skupin podle toho, kam patří, což víme z předchozí fáze. Po něm následuje operace na vyrovnání přebytků/nedostatků hráčů ve skupinách.

Vyrovnání velikostí skupin

Zde je nutnost si položit otázku, co je spravedlivé při vyrovnávání velikostí. Rovnou si můžeme prozradit, že odpověď neexistuje. Jinak bychom museli být schopni odpovědět na otázku: co je těžší, udržet se v první skupině, nebo vyhrát druhou/třetí/...? Po chvíli přemýšlení zjistíme, že to hodně záleží na počtu a úrovni hráčů, a tedy že se to liší liga od ligy. Nebude proto nepřekvapením, že se znovu obrátíme na nastavitelná pravidla.

Samotný mechanizmus vyrovnání probíhá tak, že se jde od první skupiny k poslední a vždy se vyrovná počet hráčů v ní tak, aby odpovídal cílové velikosti skupiny. Je-li ve skupině přebytek hráčů, pošlou se hráči navíc do skupiny pod ní. Kteří hráči to budou se určí dle správcem nastavených priorit pro odebírání přebytku. A naopak, je-li jich málo, hráči se doplní ze skupin pod ní. Priority pro to jaké hráče v takových případech vybrat nastavuje správce.

Dokončení

Dalším krokem, jak můžeme vidět na obrázku 5.6, je nastavení bodů skupin a bodových systémů zápasů, opět je možno využít různé strategie. Ke změnám může navíc dojít i později ve správě aktuálních kol. Přidají se hráči nad limit skupiny, tedy nastupující na základě divoké karty. Posledním procesem je skutečné vytvoření skupin a přiřazení hráčů do nich.

Ukončením poslední fáze se rozlosovávané kolo stává aktuálním. Celý proces rozlosování je dokončen tím, že kolo přestane být označeno jako nerozlosované.

První kolo ročníku je možno rozlosovat i ručně. Tím pádem neproběhne ani jedna fáze a správce si sám vytvoří skupiny a přidá do nich hráče.

Kompromisem je rozlosování poloautomatické, které ještě mezi třetí fázi a dokončení vloží ruční úpravy. Správce pak může udělat ruční zásahy nebo třeba experimentovat s různě velkými skupinami.

5.5 Speciální funkce rozlosování

Již víme, co rozlosování je a jak funguje. Podívejme se ještě na nějaké další funkce a fakta, která z toho vyplývají.

Mezi koly je možné přidávat libovolný počet hráčů, jak normálně tak na divokou kartu. Můžeme je i libovolně odebírat. Systém si poradí i se změnou standardní velikosti skupin.

V praxi se stává, že nastane před rozlosováním chyba. Třeba označíme jako vzájemnou skreč zápas, který byl odehrán, ale ani jeden ze soupeřů si nevzpomněl ho zadat. Rozlosování kola je proto možné vrátit, opravit chybu a znovu rozlosovat.

To ale sebou nese další problém. Co když jsme se tak rozhodli učinit pozdě a někteří hráči stihli odehrát zápas dle špatného rozlosování? Pokud jsou po novém rozlosování každý v jiné skupině, tak nás čeká buď vrácení chyby nebo omluva hráčům. Co když se jich ale změna nedotkla? O odehrané zápasy přijít nemusíte. Systém si takové zápasy pamatuje (pokud mu to nezakážeme) a při novém rozlosování se je pokusí obnovit.

Příchody a odchody hráčů si můžeme naplánovat dopředu. Vyhneme se tak situaci, kdy si to musíme poznamenávat někam stranou a doufat, že na to v den rozlosování nezapomeneme.

6. Popis aplikace z hlediska uživatele

Od dob, kdy autor začal vést svou první ligu, se technika rozhodně posunula kupředu. Liga začínala jednoduše, hráčů bylo pár a výsledky se zapisovali do sešitu na recepci. Jenže popularita rychle vzrostla, hráči přibyli a něco takového přestalo stačit. Centrum si proto pořídilo relativně jednoduchý, téměř textový prográmek, aby se vše ulehčilo. Nedá se říci, že by neposloužil, naopak ušetřil spoustu práce a centrum za něj bylo vděčné. Jenže nároky ligy dál rostli, situace se vyvíjela, technika pokročila a limity se brzo ukázali. A tak se pomalu zrodila myšlenka udělat moderní systém s vyšší mírou obecnosti tak, aby dokázal vyhovět požadavkům více center. A rozhodně by měl práci usnadnit a ne ji přidělávat.

Silnou předností naší aplikace je její nastavitelnost a bohatost funkcí. Velké množství úkonů si v ní může administrátor udělat sám. Aplikaci proto není třeba neustále modifikovat, když se například změní pravidla pro určení hráčů ve skupině.

Například by se nemělo neustále platit správci, aby jim připravoval první kola ročníků, když plánují provozovat ligu mnoho let. My proto takové funkce rovnou dodáváme. Lepší je mít a třeba je vypnout než je nemít vůbec a pak je dlouhou dobu dodělávat.

V této kapitole se podíváme na nějaké tyto přednosti. Na rozdíl od kapitoly 5 si je ale představíme z hlediska uživatele.

6.1 Responzivní uživatelské rozhraní

Uživatelské rozhraní aplikace je nepochybně důležitá věc. Jednou z výhod sportu je, že se při něm můžeme odreagovat a vybít si svou frustraci. Byla by škoda tento efekt znegovat tím, že si hned potom zkazíme náladu při ukládání výsledků.

Problematiku dnešních uživatelských rozhraní nejlépe popisuje následující autorova zkušenost:

„Jednou jsem šel u nás v centru kolem kurtů, když slyším peprné nadávání. To se jistě u některých hráčů stává, sport dokáže rozproudit emoce. Zajímavé ale bylo to, že zvuk přicházel z vnější strany kurtu a ne z něj. Jdu se podívat, co se děje a vidím, jak se hráč pobouřeně hrabe v mobilu. Zeptám se, co se děje a je mi řečeno, že chce zadat výsledek, ale nedaří se mu tam proklikat. A už má jít hrát další zápas. Tak mu řeknu, že mu to tam zadám, ať jde hrát. Podá mi mobil a odchází. Tak to zkouším také, a teprve v tu chvíli plně docením, co myslí. Ono se to opravdu dělá dosti nepohodlně. Naštěstí vím přímou adresu, takže se vyhnu proklikávání se přes odkazy. Jenže i naklikání výsledku se mi nedaří. Po pár minutách uznám, že nebudu ztrácet čas a raději si dojdu k počítači. V době vzniku aplikace taková zařízení nebyly, tak to autorovi těžko můžeme mít za zlé. I tak mi to ujasnilo jednu věc, ovládání pomocí smartphonu je důležité a ač jsem sám fanoušek takových zařízení nebyl, musí to jít na seznam požadavků. “

Proto naše aplikace použila framework Bootsrap (viz 3.9.2) využívající responzivního design schopného přizpůsobit se obrazovkám mobilních zařízení. Přesto ale by to nemělo být úplně na úkor počítačů. Když už si uživatel pořídí velký mo-

Obrázek 6.1: Uživatelské rozhraní pro zadávání výsledků.

nitor, tak nechce používat miniaturní rozhraní, přestože za chvíli popadne tablet a bude chtít opak. Aplikace má proto dostatečně velké ovládací prvky, aby se dobře ovládala prstem, jak je vidět na obrázku 6.1. Zároveň se ale přizpůsobuje velikostem obrazovek až po přepnutí do mobilního zobrazení. Například počet sloupců zobrazených kol se mění dle šířky obrazovky (obrázek 6.2), či menu se zobrazuje jiným způsobem. O tvorbě responzivního designu webových stránek se lze dočíst v knize Responsive Web Design [10].

6.2 Lokalizovatelnost

Jak jsme si uvedli hned v úvodu, česká lokalizace ovládání je nezanedbatelný požadavek. Cílíme přece jen na lokální sport centra a ne nějaké mezinárodní organizace. A přece jen u nás v Česku stále mnoho lidí bojuje s angličtinou. Veškeré texty UI jsou proto psány v českém jazyce.

Ovšem po tolika úsilí věnovaném vývoji by bylo škoda omezit se jen na češtinu. Vždyť jedním z cílů podpora sportu a tomu je jedno, zda je provozován u nás nebo třeba v sousedním Německu.

Veškeré texty uživatelského rozhraní jsou proto uloženy v externích souborech. Díky mechanizmu výběru je možno překládat jen to, co je třeba. Pokud něco není přeloženo, automaticky se použije původní verze. Je možné překládat i pro specifické lokalizace a opět jen to co se liší. Můžeme mít tedy překlad pro angličtinu a dopsat rozlišující části pro americkou a britskou. Společnou část nám stačí napsat jen jednou. Ani by nebylo třeba velkého úsilí, kdybychom chtěli aplikaci rozšířit tak, aby umožňovala výběr jazyka za běhu.

6.3 Logování událostí

Jedna z menších otázek, kterou bylo třeba rozhodnout bylo i to, zda správce smí ukládat výsledky pouze potvrzené. Na jednu stranu by se dalo říci, že by se administrátor neměl tvářit jako uživatel. Z praxe ale můžeme říci, že to tak

Obrázek 6.2: Zobrazení ročníku ligy v responzivním designu, počet zobrazených sloupců se mění dle šířky obrazovky.

není. Není zase až tak nezvyklé, že by některý hráč požádal, aby se mu zapsal nějaký výsledek, například proto, že mu přestal fungovat počítač. Do konce kola daleko a co teď? Buď musíme počkat až do konce nebo si to někam poznamenat a doufat, že ho nezapomeneme zapsat. A nebo ho zapíšeme rovnou jako potvrzený. A tak jsme obešli potvrzovací systém. Je proto lepší mít možnost zapsat ho jako nepotvrzený. Soupeř pak má normálně právo nesouhlasit.

Podobná situace nastává při změně uživatelského účtu, když jsme požádání, provést nějaké změny za uživatele. Třeba mu změnit telefonní číslo ze starého na nové.

To jsou některé z důvodů, proč bylo do systému zavedeno logování událostí. Akce které mohli vyvolat různí uživatelé, či aplikace samotná, se ukládají do logu. Není pak problém dohledat, kdo že to kdy zadal/změnil/zrušil daný výsledek, upravil účet a podobně.

Přístup k němu má samozřejmě pouze administrátor. Ten si může u každého typu nastavit, zda ho chce sledovat. Pokud někdo jiný než je on sám danou akci provede, v aplikaci se mu zobrazí mezi sledovanými. Jelikož o akcích, které provedl sám ví, tak takové se mu tam nepřidávají. Má i možnost nechat si přitom zaslat email. Může tak být okamžitě upozorněn, když třeba uživatel způsobí konflikt zadáním rozdílného výsledku než soupeř.

Skupina 1

Bodů za skupinu : 5

Příjmení, Jméno		1	2	3	4	Bodů	Skrečí
Kvapil, Martin	1		**3:1**	3:1		6	0
Novotný, Antonín	2	**1:3**		**3:2**	3:0	7	0
Tučný, Josef	3		**2:3**			3	0
Veselý, Petr	4	0:3	2:3			0	0

Obrázek 6.3: Skupina v lize s různými druhy výsledků, zvýrazněním přihlášeného hráče. Nastaveno je zobrazení počtu bodů a skrečí.

6.4 Vizualizace výsledků

6.4.1 Skupinové výsledky

Na obrázku 6.3 můžeme vidět zobrazení výsledků jedné skupiny. Globálně je možné aplikaci nastavit, zda má zvýrazňovat přihlášeného uživatele i jaké sloupce mají být zahrnuty. Jsou možnosti jako počet bodů, poměr setů a další. Nastavení jdou provést zvlášť pro osamocenou skupinu (třeba v přehledu skupiny přihlášeného uživatele) i pro zobrazování více skupin pohromadě v rámci kola. Hráč si může nastavit i zda se mu má ukazovat pořadí hráčů, a zda se při jeho výpočtu mají zahrnout i nepotvrzené výsledky. Díky tomu tak při zadávání okamžitě vidí, jak se situace změnila.

6.4.2 Žebříček hráčů

V každém ročníku je sestaven žebříček hráčů podle bodů, které v něm hráč nasbíral. Stejně jako mnoho jiných částí naší aplikace i zde můžeme nastavovat pravidla, konkrétně jaké body se do něj mají započítávat. Podobně jako pravidla pro obhájení divoké karty i tyto jsou parametrizována. Můžeme třeba škrtnou dva nejlepší a nejhorší výsledky, počítat jen osm nejlepších a podobně. Jde jich určit i více za sebou a jsou nastavitelné zvlášť pro turnaje a ligu.

Zajímavou informací je jistě i to, jak se stav vyvíjel v průběhu času. I to je možné. Zobrazit stav lze po každé změně neboli po každém kole a turnaji. Samozřejmostí je i zobrazení detailů kolik bodů hráč obdržel při jaké akci. Několik prvních hráčů žebříčku můžeme vidět na obrázku 6.4.

6.4.3 Statistiky

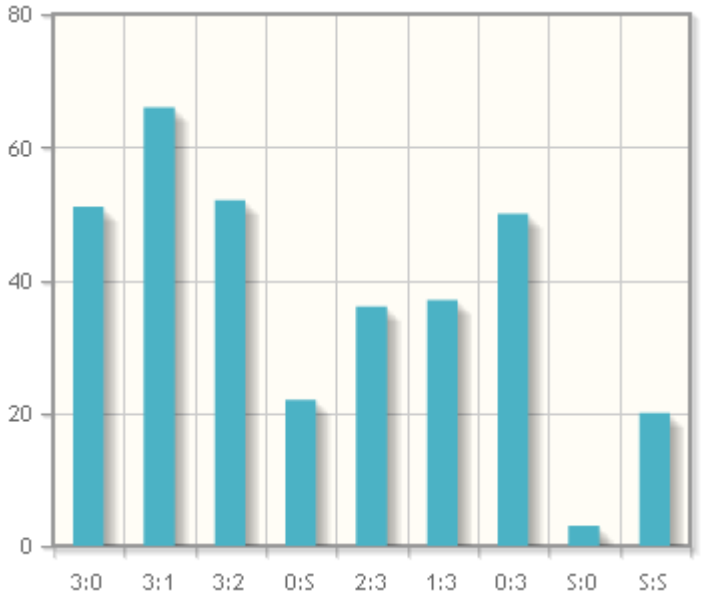
Uživatelům můžou přijít vhod i různé statistiky pro sledování svého výkonu i výkonů svých soupeřů. Jeden z grafů se statistikou můžeme vidět na obrázku 6.5.

Díky oddělenému systému zápasů se mohou porovnávat výsledky skrze různé ročníky i ligy. Co si třeba zobrazit, který hráč má nejvíce vítězství? Vždy ale nemá smysl porovnávat vše se vším. Můžeme si proto vybrat, ze kterých kol

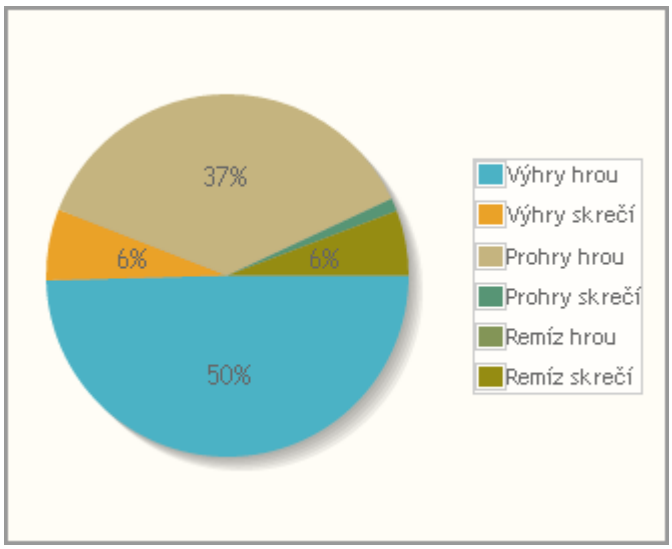
Žebříček po 9. kole

Pozice	Příjmení, Jméno	Celkem	Liga	Turnaje
1	Novoveský, Jaromír	897	698	199
2	Kracík, Bohumil	894	705	189
3	Svoboda, Jaroslav	882	780	102
4	Kulíková, Amálie	873	679	194

Obrázek 6.4: Žebříček hráčů je možné si zobrazit po jakémkoliv odehraném kole či turnaji.



Obrázek 6.5: Jedna z možných statistik je sloupcový graf počtu výsledků vybraného hráče.



Obrázek 6.6: Další možnou statistikou je koláčový graf poměrů výher, proher a remíz vybraného hráče.

se má statistika počítat. Lze se tak zaměřit jen na vybrané ligy nebo třeba jen na současný ročník.

Uživatel si zobrazí své statistiky výher a proher (lze vidět na obrázku 6.6), podívá se, jak si stojí vůči konkrétnímu hráči nebo zjistí, koho nejčastěji porazil a s kým má nejvíce proher a případně tomu přizpůsobí svůj trénink. A ještě si to omezí zobrazení na vybrané období, aby se mu do něj nezapočítávala doba, kdy začínal a ještě neuměl dobře hrát. Takové období o jeho současné formě nic neříká.

Centrum pak může sledovat vývoj počtu účastníků lig, nebo si zjistit hráče s největším počtem neodehraných zápasů.

6.5 Ostatní

Aplikace má přirozeně řadu dalších funkcí a možností. Pro detailnější pohled je možno nahlédnout do uživatelské dokumentace.

Je například možné si zobrazit historii vybraného hráče ve všech kolech a turnajích, podívat se na stará kola. Uživatel si může zapnout zobrazení pořadí ve skupině a nastavit si, aby se mu počítali nepotvrzené výsledky jako potvrzené a mít tak okamžitý přehled o své situaci.

Jelikož některé ligy již mají nějakou historii za sebou, mohla jejími řadami projít slušná řádka hráčů. Někteří už třeba několik let nemají s ligou nic společného. Takoví by asi nestáli o to, aby jim najednou emailovou schránku začala plnit nevyžádaná pošta jenom proto, že se nějaký hráč přihlásil do ligy, okopíroval si kontakty a zase z ní odešel. Hráčům se proto zobrazují jen kontakty na hráče, co hrají ve stejné lize. Pokud někdo skončí, kontakt na něj se hráčům přestane zobrazovat.

Pro pohodlí uživatelů a zjednodušení správy soutěže může administrátor vytvořit soupisku pro přihlašování do vybraných kol a turnajů. Zvolí si termín, kdy se hráči mohou hlásit, zda je třeba jeho potvrzení a zda se potom mohou sa-

mi odhlásit. Je možné zveřejnit počet přihlášených i seznam jmen, případně celé přihlašování zobrazit veřejnosti a ne jen hráčům. Může si do seznamu přidávat i anonymní hráče, jež se ligy neúčastní.

Stejně tak může vytvořit nějakou anketu, ve které můžou hráči ligy hlasovat. Počet možností je libovolný a u každé lze nastavit, zda k ní má uživatel připsat nějaký komentář. Opět má termín pro hlasování, jdou nastavit zobrazení poměru či přímo počtu hlasů i to, zda výsledky vidí i anonymní uživatel.

Aplikaci lze globálně nastavit, zda se mají veřejnosti zobrazovat i nepotvrzené a konfliktní výsledky, nebo třeba jaké položky se mají zobrazovat u skupin.

Shrňme si nejzajímavější vlastnosti aplikace:

- Responsive UI
- Lokalizovatelnost
- Logování událostí včetně nastavitelné notifikace u vybraných událostí
- Bohatý přehled výsledků a statistik
- Libovolné množství lig, ročníků, kol a turnajů, každé s vlastním nastavením
- Libovolné množství přicházejících a odcházejících hráčů, včetně divokých karet i naplánování do budoucnosti
- Bohatá nastavitelnost rozlosování (určení pořadí hráčů, obhájení divoké karty,...) a pravidel počítání bodů, včetně funkce pro návrat rozlosování či jeho ruční vytvoření
- Nastavitelné herní a bodové systémy zápasů
- Systém potvrzování výsledků
- Možnost přihlašování se do kol a turnajů v rámci aplikace
- Možnost vytváření anket

7. Závěr

Vytvořili jsme aplikaci, pomocí které mohou sportovní centra spravovat své soutěže jednotlivců. Definovali jsme si model ligové soutěže a věcí s ní spojených a určili jsme stavy v jakých se mohou nacházet. Představili jsme si problémy s tím spojené a ukázali jsme si, jak jsme se s nimi vypořádali. Vznikl tak znovupoužitelný silně přizpůsobitelný systém. Pomocí šikovného návrhu jsme se připravili i na případné změny tak, abychom s nimi pak měli co možná nejmenší problémy. Díky jednoduchému ovládání, modernímu uživatelskému rozhraní a možnosti aplikaci lokalizovat i díky zmíněné přizpůsobitelnosti jsme zajistili možnost cílit na co nejširší publikum.

Četné funkce by měli uživatelům přinést pohodlí a možnost, jak své výsledky vyhodnotit, upravit na jejich základě svůj trénink a zlepšit se tak. Administrátorům ulehčuje a zrychluje práci, aby nebyla veškerá energie spotřebována při správě a mohla tak být cílena k dalšímu rozvoji ligy. Sportovnímu centru aplikace nabízí způsob, jak si s malým úsilím zvýšit svou klientelu založením vlastní lokální ligy.

Třešničkou na dortu je fakt, že aplikace byla s úspěchem zavedena do praxe již při svém vývoji a svůj cíl tak již plní, místo aby se na ní pouze snášel prach a jediným přínosem by byly četné znalosti, které jsme při vývoji tohoto díla museli získat. Přineslo to i tu výhodu, že aplikace byla otestována skutečnými nároky, na které mohla i reagovat a nestala se tak dílem teoreticky zajímavým, ale v praxi nepoužitelným.

Přesto asi jako u každého díla zůstává prostor pro další růst. Je možné i dále reagovat na rychle se vyvíjející svět chytrých telefonů a vytvořit přídavnou aplikaci pro zařízení s Androidem, iOS či Windows nebo přidat další druhy statistik.

Seznam použité literatury

[1] Scott Chacon and Ben Straub. *Pro Git*. Apress, 2 edition, 2014.

[2] John Deacon. Model-view-controller (mvc) architecture, May 2009.

[3] Marie Demlová. Diskrétní matematika a logika pro km, November 2010.

[4] Barry Dorrans. *Beginning ASP.NET Security*. John Wiley & Sons, Ltd, 2010.

[5] Roger Jenings. *Professional ADO.NET 3.5 with LINQ and Entity Framework*. Wiley Publishing, Inc., 2009.

[6] Glenn E. Krasner and Stephen T. Pope. A cookbook for using the model-view-controller user interface paradigm in smalltalk-80. *Journal of Object-Oriented Programming*, pages 26–49, 1988.

[7] Matthew MacDonald. *Beginning ASP.NET 3.5 in C# 2008: From Novice to Professional*. APress L. P, 2 edition, 2007.

[8] Matthew MacDonald and Mario Szpuszta. *ASP.NET 2.0 a C# - tvorba dynamických stránek profesionálně*. Zoner Press, 2006.

[9] Alex Mackey. *Introducing .NET 4.0 With Visual Studio 2010*. Paul Manning, 2010.

[10] Ethan Marcotte. *Responsive Web Design*. A Book Apart, 2 edition, 2014.

[11] Fabrice Marguerie, Steve Eichert, and Jim Wooley. *LINQ in Action*. Manning Publications, 2008.

[12] Jeffrey Palermo, Jimmy Bogard, Eric Hexter, Matthew Hinze, and Jeremy Skinner. *ASP.NET MVC 4 in Action*. Manning Publications Co., 2012.

[13] José Rolando Guay Paz. *Beginning ASP.NET MVC 4*. Paul Manning, 2013.

[14] Paolo Pialorsi and Marco Russo. *Programming Microsoft® LINQ in Microsoft .NET Framework 4*. O'Reilly Media, Inc., 2010.

[15] Danny Ryan and Tommy Ryan. *ASP.NET Your visual blueprint for creating Web applications on the .NET framework*. Hungry Minds, Inc., 2002.

[16] Karl Swedberg and Jonathan Chaffer. *jQuery 1.4 Reference Guide*. Packt Publishing Ltd., 2010.

[17] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, pages 146–160, 1972.

Seznam tabulek

Seznam použitých zkratek

CLI	Common Language Infrastructure
CIL	Common Intermediate Language
CLR	Common Language Runtime
JIT	Just in Time
BCL	Base Class Library
FCL	Framework Class Library
ASP	Active Server Pages
LINQ	Language Integrated Query
API	Application Programming Interface
SQL	Structured Query Language
EF	Entity Framework
MVC	Model View Controller
HTML	HyperText Markup Language
WYSIWYG	What you see is what you get
DOM	Document Object Model
CSS	Cascading Style Sheets
AJAX	Asynchronous JavaScript and XML
SDK	Software development kit
URL	Uniform Resource Locator