UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Tvorba WWW aplikace s využitím relační databáze pro oddíl SK Studenec

Michal Erlebach

Bakalářská práce
2011

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2010/2011**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Michal ERLEBACH**
Osobní číslo:	**I07599**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Tvorba WWW aplikace s využitím relační databáze pro oddíl SK Studenec**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Úkolem bakalářské práce je vytvořit webový portál pro oddíl SK Studenec zahrnující redakční systém a jednoduchý systém pro správu obsahu.
Obsahem teoretické části bude popis softwarové architektury Model-View-Controler se snahou o její implementaci v praktické části bakalářské práce.
Obsahem implementační části bude vlastní realizace portálu klubu zabývajícího se fotbalem, orientačním během a alpským lyžováním. Systém pro správu obsahu umožní správu kontaktů a sponzorů oddílu, redakční systém umožní vkládání článků a krátkých informačních zpráv s možností exportu do formátů RTF a PDF. V neveřejné části aplikace bude řešena správa uživatelů a jejich přístup dle rolí. Práce bude realizována na platformě PHP a MySQL.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

***ULLMAN, Larry. PHP a MySQL : Názorný průvodce tvorbou dynamických WWW stránek. Vydání první. Brno : Computer Press, 2004. 534 s. ISBN 80-251-0063-4.**

***MEYER, Eric A. Eric Meyer o CSS - kompletní průvodce. Vydání první. Brno : Zoner Press, 2007. 560 s. ISBN 978-80-86815-64-0.**

***WELLING, Luke; THOMSON, Laura. PHP and MySQL Web Development. Third Edition. Indianapolis : Sams Publishing, 2005. 946 s. ISBN 0-672-32672-8.**

Vedoucí bakalářské práce: **RNDr. Iva Rulicová**
Katedra informačních technologií

Datum zadání bakalářské práce: **17. prosince 2010**

Termín odevzdání bakalářské práce: **13. května 2011**

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2011

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

Ve Studenci dne 21. 4. 2011 Michal Erlebach

Anotace

Tato bakalářská práce je zaměřena na vývoj webové aplikace pro oddíl SK Studenec. První část práce je čistě teoretická a zabývá se softwarovou architekturou Model-View-Controller. Konkrétně je zde objasněn princip, vlastnosti a varianty MVC. Koncept MVC je poté využit při realizaci portálu SK Studenec, které je věnována druhá část této práce. Součástí praktické části je také popis řešení konkrétních situací při vývoji webové aplikace jako například návrh databáze, vzhled a rozvržení stránek nebo systém uživatelských rolí.

Klíčová slova

MVC, model-view-controller, MVP, model-view-presenter, softwarová architektura, PHP, MySQL, webová aplikace, redakční systém, CMS, SK Studenec

Title

WWW application with usage of relational database for SK Studenec club

Annotation

This bachelor thesis focused on web application development for SK Studenec club. The first part is purely theoretical and deals with Model-View-Controller software architecture. Specifically is here explanation of the MVC principles, features and variations. The MVC concept is then used for the implementation SK Studenec web sites, which is devoted to the second part of this work. The practical part describes also specific situations in developing web application such as database design, web sites layout and design or system of user roles.

Keywords

MVC, model-view-controller, MVP, model-view-presenter, software architecture, PHP, MySQL, web application, content management system, CMS, SK Studenec

Obsah

Seznam zkratek

MVC	Model View Controller
MVP	Model View Presenter
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
HTML	HyperText Markup Language
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language
PHP	PHP Hypertext Preprocessor
ASP	Active Server Pages
URL	Uniform Resource Locator
CMS	Content Management System
ACL	Access Control List
SQL	Structured Query Language

Seznam obrázků

1 Úvod

Rychlý vývoj webových technologií a nástup nových moderních webových služeb, kde se tvůrci obsahu stávají samotní uživatelé, klade nejen vysoké nároky na znalosti vývojářů, ale i nemalé požadavky na přehlednou strukturu a organizaci aplikace. Přechod od statických webových stránek k dynamickým webovým aplikacím má za následek nárůst velikosti zdrojových kódů více jak o desetinásobek. Počet řádků zdrojového kódu roste také díky rozmanitosti koncových zařízení, jako jsou v dnešní době oblíbené smartphony a tablety. Vývojový tým webové aplikace s ambicemi dlouhodobě udržitelného vývoje se také s přibývajícími požadavky koncových uživatelů rozrostl ze stádia one-man-show na pár desítek vývojářů. Z výše ve stručnosti zmíněných avšak neméně podstatných důvodů by se daly shrnout následující požadavky na vývoj moderní webové aplikace: modularita, přehlednost a rozšiřitelnost. Docílení těchto tří bodů by nám měla zajistit vhodná volba a správná implementace architektury webové aplikace, které bude věnována, konkrétně architektuře MVC (model-view-controller), tato bakalářská práce.

Prvotním námětem pro výběr toho tématu byla žádost od představenstva sportovního oddílu SK Studenec na vytvoření prozatím jednoduchého redakčního systému s možností budoucího rozšíření o další funkcionalitu například přihlašovacího systému. Splnění požadavků na jednoduchost a rozšiřitelnost systému se tedy stalo hlavním úkolem.

Při hledání vhodného řešení jsem nejdříve nakoukl do zdrojových kódů vlastního projektu nazvaného Fotogalerie, který jsem zpracovával na předmět Návrh a tvorba www stránek. I když byl kód důkladně obklopen obsáhlými komentáři, schopnost orientovat se v procesech (zachycení a zpracování požadavků klienta, zpracování odpovědi na požadavek) nebyla valná. O možnostech jednoduchého rozšíření o libovolnou funkcionalitu nemohla být ani řeč. Toto zděšení mě vedlo k tomu, poohlédnout se po moderních způsobech programování rozsáhlejších webových aplikací. Jakým směrem se ubírat mne navedla krátká zmínka o MVC na přednášce z výše uvedeného předmětu.

Cílem této bakalářské práce je tedy objasnit princip architektury MVC a následně realizovat tyto principy při tvorbě webové aplikace pro oddíl SK Studenec.

První kapitola je věnovaná teoretické části, kde se ve zkratce věnuji charakteristice, principům a možnostem využití této architektury. Dále jsou zde shrnuty výhody a nevýhody použití architektury MVC a její způsoby implementace. Z hlediska implementace jsou zde také zmíněny rozdíly mezi MVC při tvorbě desktopových a webových aplikací.

Druhá kapitola je už zaměřena na praktickou část této práce. Nejprve ukazuji využití architektury MVC při tvorbě portálu pro oddíl SK Studenec. Popisuji zde vlastnosti

a vztahy mezi komponentami ve vlastním řešení MVC. Součástí popisu jsou i krátké ukázky zdrojových kódů pro lepší objasnění situace a některé diagramy, které by měly posloužit k lepšímu pochopení a zjednodušit pohled na fungování mého postupu. Zbylá část druhé kapitoly se plně věnuje obecné realizaci portálu pro oddíl SK Studenec. Jsou zde podrobně popsány požadavky na konkrétní funkce, jako je například správa článků, kontaktů nebo uživatelů. Dále jsou zde popsány uživatelské role v rámci autorizačního systému nebo adresářová struktura aplikace. Část je také věnována řešení databáze a přístupu k ní.

2 Architektura Model-View-Controller

Sousloví Model-View-Controller (MVC) je nečastěji spojováno se způsobem návrhu aplikace. Jde o softwarovou architekturu, která rozděluje aplikaci na tři na sobě nezávislé části [9]:

- datovou vrstvu – model,
- uživatelské rozhraní – view,
- a aplikační logiku – controller.

Jednotlivé komponenty mezi sebou spolupracují, každá má v rámci aplikace jinou úlohu. Lze je proto modifikovat samostatně bez ovlivnění ostatních částí. Podrobně se vlastnostem, úkolům a závislostem mezi jednotlivými komponentami věnuji v následující kapitole.

Názorů a variant, jak má správně MVC fungovat, je k nalezení mnoho, avšak definice, která by přesně vymezila vlastnosti tří zmíněných komponent a závislosti mezi nimi, neexistuje. To vše souvisí i s tím, že není jednoznačné, jak tento princip pro tvorbu aplikací označit. Je otázkou, zda se jedná o architekturu, návrhový vzor, koncept nebo obecný přístup pro vývoj složitějších aplikací. Já v této práci používám a budu používat spojení *architektura MVC*. Pod slovem *architektura* aplikace si lze představit způsob rozvržení aplikace do logických částí, stanovení vlastností těchto částí a vzájemných vazeb mezi nimi. Toto rozvržení by mělo probíhat na abstraktní úrovni, aby bylo použitelné pro jakékoliv konkrétní implementace.

2.1 Princip MVC

Jednotlivé kódy aplikací (nejen webových) lze ve zjednodušení zařadit do oblastí, které mají společné úkoly. Do jedné oblasti je možné zahrnout business logiku aplikace, strukturu dat a přístup k nim – **model**. Další oblast zdrojových kódů by mohla mít na starosti výstup, tedy zobrazení dat v uživatelsky přívětivé podobě – **view**. Poslední a neméně důležitou kategorií pro odlišení kódů je oblast, která zahrnuje kódy, jež mají za úkol řídit chování celé aplikace – **controller**.

Princip MVC vyžaduje nejen striktní organizaci zdrojových kódů na tyto oblasti, ale i nutnost izolovat je do samostatných a nezávislých modulů. Splnění tohoto principu napomáhá k udržení přehlednosti kódu. Popišme si jednotlivé moduly obecné architektury MVC podrobněji.

2.1.1 Model

Model je z pohledu celé architektury chápan jako celek, avšak při realizaci je tvořen množinou funkcí nebo tříd. Tyto funkce resp. třídy vykonávají především operace s daty (načítaní, zpracování, ukládání). Součástí modelu je ale také, kromě zajištění přístupu k datům (např. komunikace s databází) a manipulace s nimi, veškerá business logika, která je zodpovědná za chování celé aplikace.

Model ze svého pohledu neví nic o view a controlleru. Správně vytvořený model by měl tedy být použitelný bez jakýchkoliv změn pro jakýkoliv typ aplikace, ať už se jedná o aplikaci pro příkazový řádek, desktopovou okenní aplikaci nebo jako v mém případě aplikaci určenou pro web.

Model také odstiňuje způsob uložení dat. Pro view a pro controller není důležité, kde jsou data uložena ani v jaké podobě (databáze, XML soubory apod.).

2.1.2 View

View neboli pohled prezentuje data, která zastupuje model. Má na starosti výstup aplikace v grafické, textové či jiné podobě.

Pohled přistupuje k datům modelu přímo přes rozhraní, které model nabízí (pohled vytahuje data z modelu) nebo je naplňován daty pomocí controlleru, tzv. *Passive view* [2]. Data modelu mohou být zobrazována v různé podobě. V případě webové aplikace lze data prezentovat například ve formátu některého ze značkovacích jazyků, jako jsou HTML, XHTML či XML, nebo v podobě prostého textu.

Konkrétní vlastnosti view jsou ale závislé na typu aplikace. Rozdíl mezi webovým a widgetovým uživatelským rozhraním a jeho vliv na view rozeberu v podkapitole níže, kde se věnuji jednotlivým variantám a odlišnostem MVC.

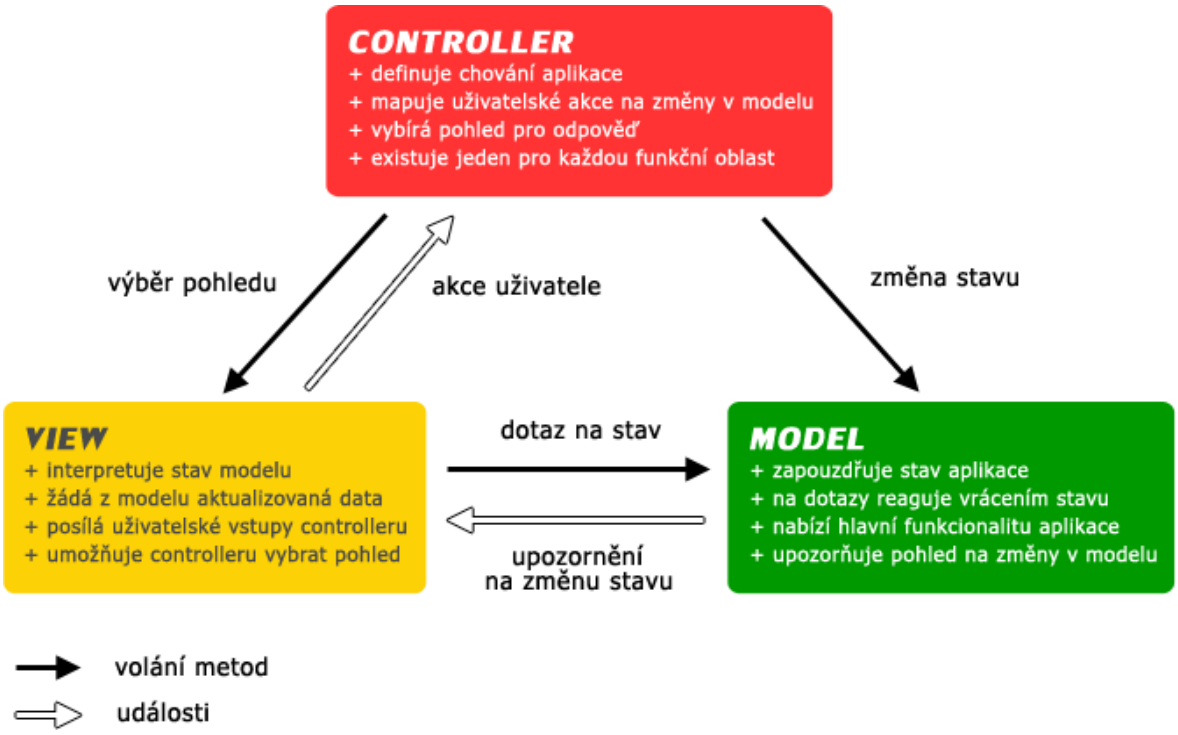
2.1.3 Controller

Controller, česky řadič, je poslední a na pochopení asi nejsložitější část architektury MVC. Už ze samotného názvu tohoto modulu vyplývá, že bude plnit funkci jakési řídící jednotky a to mezi komponentami view a model. Jeho úkoly stejně jako v případě pohledu se však také liší podle jednotlivých variant MVC. Zjednodušeně lze ale říci, že controller reaguje na požadavky uživatele a zabezpečuje změny v pohledu nebo v modelu. Rozhoduje o tom, jaké procedury se v rámci modelu vykonají. Následně podle stavu modelu a potřeb uživatele volí patřičný pohled.

V případě webové aplikace je řadič reprezentován řadou spustitelných skriptů, ke kterým aplikace přistupuje pomocí metod GET nebo POST.

2.1.4 Návaznost modulů MVC

Architektura MVC nevymezuje pouze vlastnosti jednotlivých komponent (modelu, view a controlleru), ale také definuje některé přesné, jiné volnější vztahy mezi nimi. Následující obrázek ilustruje obecné souvislosti mezi prvky architektury MVC [8].



Obrázek 1 – Obecné vztahy mezi komponentami MVC

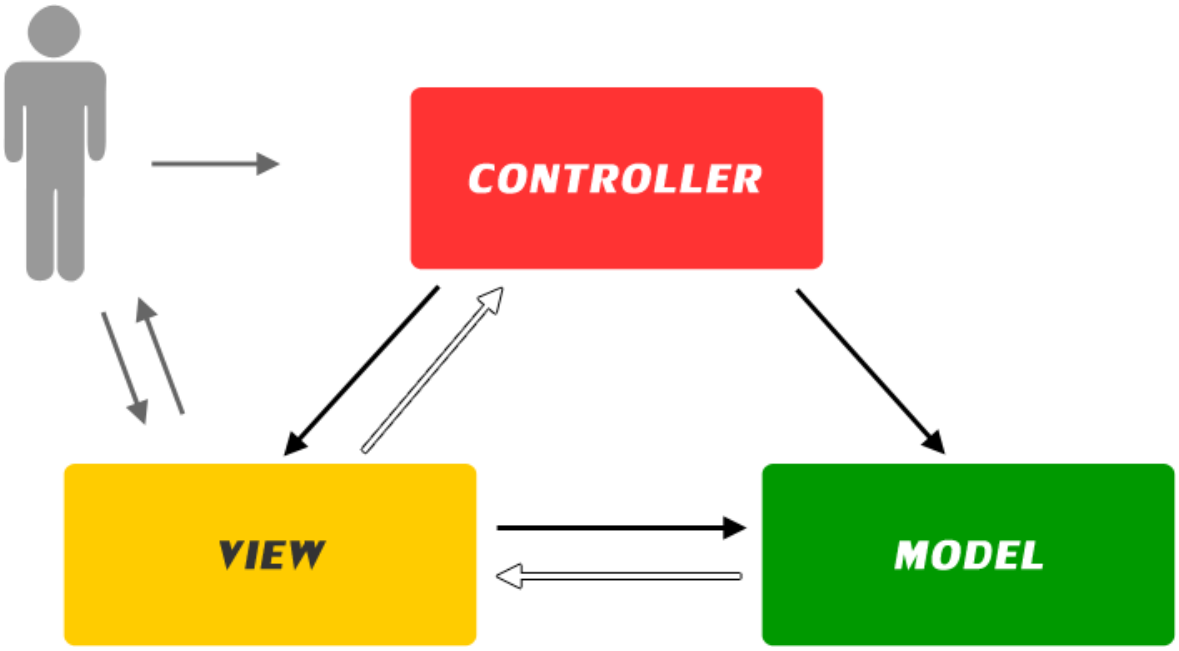
Průběh komunikace mezi komponentami je závislý na způsobu realizace MVC konceptu. Obecný postup lze shrnout do následujících sedmi bodů:

1. V uživatelském rozhraní je uživatelem vykonána jakákoliv akce. Příkladem může být například stisknutí tlačítka nebo pohyb myši v případě klasické aplikace, kliknutí na odkaz či odeslání formuláře v případě aplikace webové.
2. Prostřednictvím view nebo přímo je o vyvolané akci obeznámen controller, který na základě použitého objektu v uživatelském rozhraní (desktop) respektive jména skriptu (web) rozhoduje o akcích provedených na modelu. Společně s identifikací použitého ovládacího prvku nebo názvu žádaného skriptu mohou být controlleru předány další uživatelem definované parametry nebo data.
3. Na základě vyhodnocení obdržených dat volá řadič metody modelu, kterému také předává potřebná data. Mimo to controller vybírá popřípadě mění pohled, žádá-li si to situace.

4. Model provede potřebnou hlavní funkcionalitu (doménovou logiku) aplikace a aktualizuje svá data. Může měnit například záznamy uložené v databázi nebo mít na starosti uchování perzistentních dat mezi jednotlivými požadavky.
5. V základním pojetí architektury MVC oznamuje model přímo view změnu svého stavu. V některých variantách MVC předává pohledu informaci o změně stavu modelu řadič.
6. Jakmile pohled obdrží pokyn k zobrazení, aktualizuje svá data z modelu a prezentuje je uživateli. Model je na view nezávislý, tedy o view nic neví. Proto, aby byl view schopný přijímat notifikace o změně stavu modelu, je nutné, aby se u modelu zaregistroval jako příjemce této události. Tuto funkcionalitu nabízí návrhový vzor *observer* (pozorovatel) [6].
7. Tímto je cyklus ukončen. Opakování celého cyklu nastává v případě provedení další akce uživatelem.

2.1.5 Interakce s uživatelem

Po objasnění všech vztahů mezi jednotlivými částmi architektury MVC zbývá výše ilustrovaný obrázek doplnit o interakci s uživatelem.



Obrázek 2 – Interakce konceptu MVC s uživatelem

Z obrázku je patrné, že výstup aplikace postavené na architektuře MVC probíhá vždy prostřednictvím modulu view. S uživatelským vstup je to složitější. Možnosti vstupu do aplikace jsou závislé na jejím typu. Typy aplikací, které pohlížejí na MVC odlišně, lze rozdělit na aplikace (systémy) widgetové a „newidgetové“. Vzhledem k tomu počítá

koncept MVC s možností uživatelského vstupu přes komponentu view nebo přes komponentu controller. Uveďme si rozdíly mezi těmito dvěma způsoby [2]:

- Prvním odlišným způsobem pohledu na uživatelský vstup z hlediska MVC, jsou tzv. widgetové aplikace, jenž jsou založeny na známých komponentových frameworcích.
  U těchto komponentových systémů je uživatelský vstup zpracováván modulem view. Ovládací prvky v rámci pohledu umí vstup od uživatele samy zachytit a vyhodnotit.
  Mezi widgetové aplikace lze typicky zařadit desktopové okenní GUI aplikace založené na knihovnách Swing (Java) nebo Qt (C++). Tyto komponentové systémy je také možné nalézt mezi různými webovými frameworky, jako jsou například Silverlight, Flex nebo ASP.NET Web Forms.
- Druhou možností ošetření uživatelského vstupu je pomocí řadiče. Toto řešení je používané především u „newidgetových" aplikací, kde se žádné komponenty nevyskytují. Typický je tento způsob zpracování vstupu pro webové aplikace vyvíjené pomocí programovacích jazyků, které nenabízejí žádný komponentový systém, například PHP.

Podkapitolou o vztazích mezi modelem MVC a uživatelem jsem ukončil popis obecného principu architektury MVC. Je nutné dodat, že tento pohled na fungování není jediný. Existuje několik variant, které se liší především v názoru na vlastnosti řadiče. Rozdíly a způsoby konkrétních implementací architektury MVC popíšu v následující kapitole.

2.2 Variace MVC

Princip fungování architektury MVC, jak jsme si ho představili v předchozí kapitole, vychází z původního konceptu z konce sedmdesátých let minulého století [5]. V době sálových počítačů byl vstup a výstup programu oddělený. To znamená, že odpovídal situaci, kdy je uživatelský vstup zpracováván controllerem a výstup programu prezentován pomocí view.

Vývoj programovacích jazyků a operačních systémů měl za následek vznik grafických, uživatelsky přívětivých rozhraní (GUI), která známe dnes především z desktopových aplikací. Tento způsob tvorby okenních aplikací se vyznačuje tzv. komponentovou technologií, kde každý ovládací prvek v rámci aplikace umí nejen reagovat na události vyvolané uživatelem ale i udržovat si svůj vlastní stav. Díky tomuto vývoji se aplikační logika úzce svázala s uživatelským rozhraním. Pro chod architektury MVC to mělo zásadní vliv. Uživatelský vstup není totiž ošetřován řadičem, nýbrž ovládacími prvky v rámci modulu view. Toto pojetí, tolik typické pro widgetové aplikace,

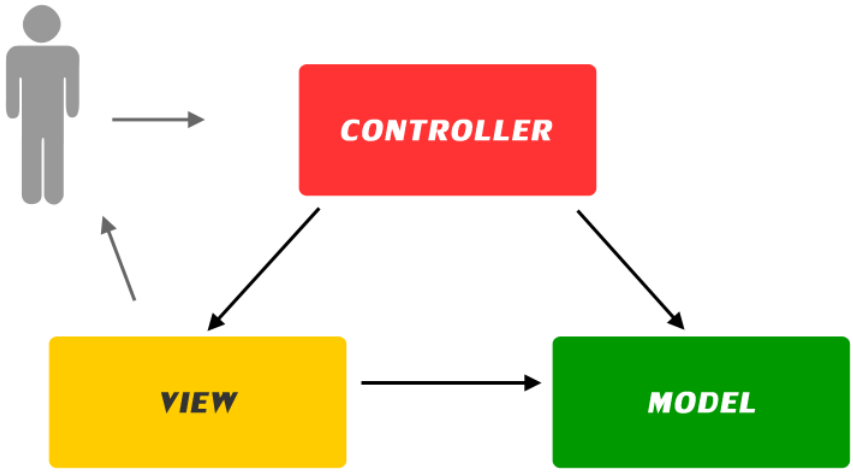
jimž byla také věnována předchozí kapitola, dostalo své specifické označení MVP (Model-View-Presenter).

Nástupem webových aplikací opět ožil původní koncept MVC s nezávislým vstupem a výstupem. Požadavek uživatele, tady vstup, je realizován odesláním příslušné URL (Uniform Resource Locator) na server. Odpověď na požadavek přichází uživateli v podobě vygenerovaného výstupu ve formátu (X)HTML či XML.

Náročnost webových aplikací měla za následek vznik nových technologií, jako například ASP .NET, které se snaží po vzoru desktopových platforem přiblížit klasickým komponentovým aplikacím. Komponenty v tomto pojetí ošetřují akce vykonané uživatelem a udržují si stav mezi jednotlivými HTTP požadavky. Tyto technologie nabízející programování webových stránek pomocí ovládacích prvků, jimiž sjednocují vstup a výstup, se opět přiklánějí k architektuře MVP.

2.2.1 Varianta Model-View-Controller

Varianta MVC je v dnešní době typická především pro webové aplikace. MVC implementované na webu se od původního konceptu této architektury příliš neliší. Pro porovnání s nadcházející variantou MVP ilustruji zjednodušené grafické schéma znázorňující návaznost pohledu, řadiče a modelu v rámci varianty MVC.



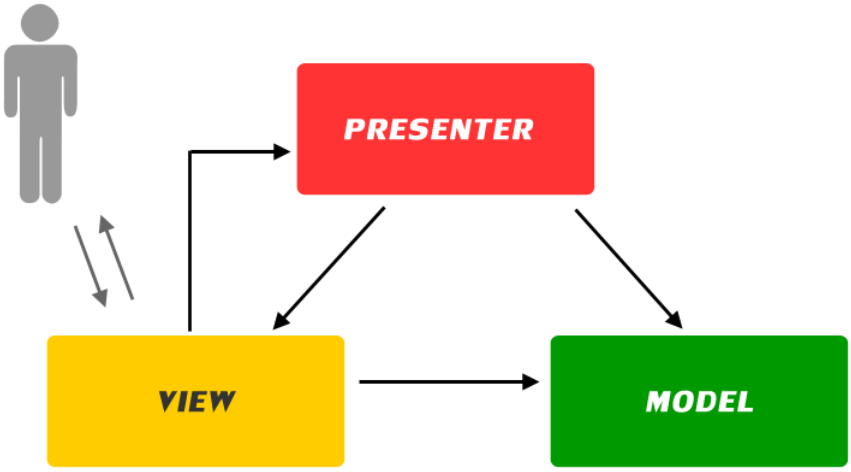
Obrázek 3 – Návaznost komponent a interakce s uživatelem v MVC

2.2.2 Varianta Model-View-Presenter

Pojetí Model-View-Presenter (MVP) se hodí do prostředí komponentových systémů. Neznamená to ovšem, že se tato sesterská varianta architektury MVC vyskytuje pouze v aplikacích určených pro desktop. Některé moderní webové technologie používají tento princip také. Kromě výše zmíněné technologie od firmy Microsoft (ASP .NET Web

Forms) lze zákonitosti MVP sledovat také u takzvaných RIA (Rich Internet Application) aplikací, jako jsou Adobe Flex, Microsoft Silverlight nebo JavaFX [2].

Obrázek ilustruje oproti architektuře MVC drobné změny, které jsou ovšem pro fungování tohoto konceptu podstatné.



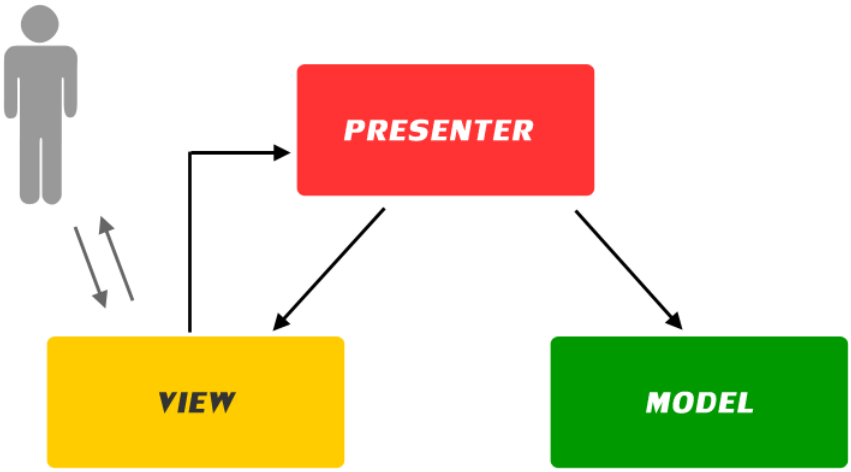
Obrázek 4 – Návaznost komponent a interakce s uživatelem v MVP

Následující výčet obsahuje několik významných rozdílů mezi architekturou MVC a MVP, které jsou z obrázku více či méně patrné:

- Presenter podobně jako v případě MVC ovládá model. Ale na rozdíl od controlleru obsahuje kromě aplikační logiky i logiku prezentační.
- Uživatelský vstup je tentokrát zpracováván komponentou view. Ovládací prvek ve view deleguje uživatelské akce tak, že spouští nějakou metodu v presenteru.
- Spolupráce mezi pohledem a presenterem je v tomto případě intenzivnější.
- Jelikož se o vstup do aplikace stará pohled, je důvodem izolace presenteru od view jeho snazší údržba, než kdyby byl součástí view.

Specifickou variantou architektury MVP je koncept *Passive view [2]*. Tento pohled na chápání MVP zde uvádím proto, že moje praktická implementace MVC vychází právě z tohoto modelu.

Koncept *Passive view* je shodný s modelem MVP až na neexistenci vazby mezi pohledem a modelem. Data z modelu nejsou v tomto případě přímo vytahována komponentou view, ale jsou předávána pohledu prostřednictvím presenteru. Pohled je tedy na modelu zcela nezávislý. Presenter má nyní kromě aplikační a prezentační logiky povinnost náležitě synchronizovat komponenty view a model. Schéma *Passive view* vypadá takto:



Obrázek 5 – Koncept Passive View

2.3 Hodnocení MVC

Rozhodování, zda architekturu MVC respektive MVP v aplikaci použít, je jednoduché. Výhody rozvržení aplikace na tři výše zmíněné části ve velké míře převažují nad jejími nevýhodami.

2.3.1 Výhody

Výhody implementace architektury MVC v aplikaci oproti běžnému stylu programování („vše v jednom“) jsou následující [9], [1], [10].

- **Znovu použitelnost zdrojového kódu, který je součástí modelu.** Oddělení modelu a pohledu umožňuje využívat schopnosti a logiku modelu při jakémkoliv zobrazení. Jelikož je model nezávislý na pohledu a controlleru, součásti komponenty modelu se snáze implementují, testují a udržují.
- **Snadná rozšiřitelnost a modularita aplikace.** Počty i obsah controllerů a pohledů může bez ovlivnění chodu aplikace růst, stejně jako doménová logika v modelu. Funkcionalitu komponent je možné využívat do té doby, dokud je zachováno jejich společné rozhraní.
- **Čistota návrhu.** Architektura MVC (MVP) zvyšuje přehlednost zdrojového kódu a minimalizuje jeho duplicity. Struktura takto koncipované aplikace zlepšuje také orientaci vývojářů v ní.
- **Snadnější podpora pro nové typy klientů.** Pro podporu nových uživatelů aplikace stačí vytvořit nové pohledy případně i specifický řadič při odlišných vstupních požadavcích. Nový pohled nebo řadič bude však přistupovat k nezměněnému modelu, přes stejné rozhraní [9].

- **Nezávislý vývoj komponent.** Prvky architektury mohou být programovány skupinou vývojářů nezávisle na sobě. Pro jednotlivé vývojové týmy je důležitá a podstatná znalost rozhraní mezi modulem, view a controllerem [9].

2.3.2 Nevýhody

Koncept MVC znovu ožil jako důsledek složité orientace v rozsáhlých aplikacích, které mají nepřehlednou stavbu. Snaží se eliminovat nedostatky těchto aplikací zpřehledněním jejich struktury, rozdělením aplikace na části s podobným zaměřením, definováním jejich odpovědností a procesů mezi nimi. Z tohoto lze vyvodit následující nevýhody architektury MVC resp. MVP:

- nárůst zdrojového kódu,
- potřeba vyvinout vlastní řešení MVC, pokud daná vývojová platforma žádné vestavěné řešení nenabízí,
- nutnost pochopit filozofii MVC pro správnou implementaci v systému.

3 Realizace webové aplikace pro oddíl SK Studenec

Ze začátku bych chtěl zmínit, že Sportovní klub Studenec je neziskové sdružení s počtem 1 až 5 zaměstnanců. Jeho snahou je provozovat oddílové webové stránky, sloužící k informování jak členů oddílu, tak široké veřejnosti o jeho aktivitách. Oddílový web tedy neslouží ke komerčním účelům, a proto jsou na jeho provoz vyčleněny minimální prostředky. Pro jeho vývoj jsou použity veřejně dostupné, bezplatné technologie, jako je PHP (skriptovací jazyk, určený především pro tvorbu dynamických webových stránek) a MySQL (databázový systém pro správu relační databáze). Obě tyto technologie podporuje mimo jiné také společnost SHOSTING, u které je zřízen webový hosting pro chod aplikace v reálném prostředí.

Tato kapitola navazuje na úvodní teoretickou část této práce. Věnuji se zde především praktické realizaci aplikace. Kapitola je rozdělena do dvou hlavních částí. V první části popisuji postup při návrhu struktury aplikace a také samotnou implementaci vlastní architektury MVC. Druhá kapitola dokresluje pohled na celkovou aplikaci. Je v ní podrobněji rozebrána hlavní funkcionalita aplikace, vlastnosti a typy uživatelských rolí nebo návrh a popis databáze.

Pro úplnost uvádím výpis technologií a prostředků, které jsem při vývoji použil.

- Technologie určené k vývoji dynamických webových stránek:
  - XHTML,
  - CSS,
  - Javascript,
  - PHP ve verzi 5.2.

- Vývojová prostředí:
  - NetBeans IDE 6.9,
  - MySQL Workbench 5.2,
  - phpMyAdmin 3.2.1.

- Webový server:
  - Apache 2.2.

3.1 Implementace architektury MVC

V úvodu bakalářské práce jsem zmínil důvody, které mě vedly k tomu, abych se zamyslel nad celkovou strukturou webové aplikace, kterou vytvářím a spravuji pro oddíl SK Studenec. Hlavní požadavky představitelů oddílu byly:

- jednoduchost,

- modularita, tzn. možnost přidání a odebrání libovolné funkcionality,
- a přehlednost.

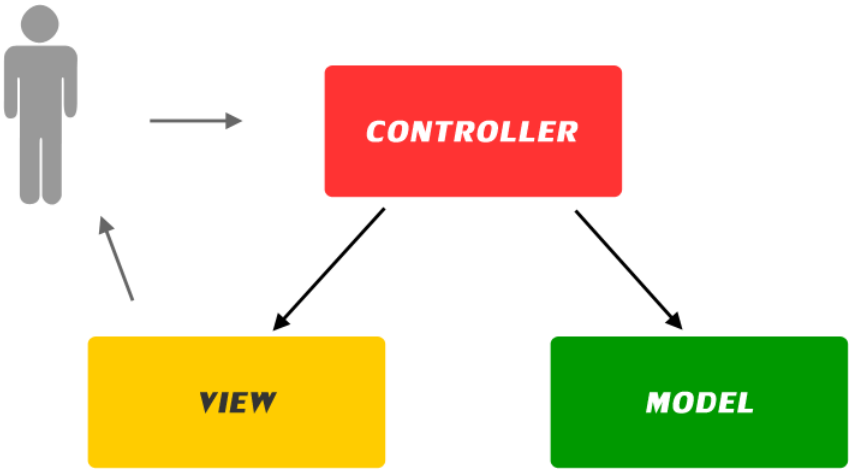
I když jsou požadavky chápány především z pohledu koncového uživatele, úzce souvisejí s návrhem a stavbou aplikace. Pro zajištění výše zmíněných nároků na provoz aplikace lze využít vlastností objektově orientovaného programování, které je podporováno v PHP od páté verze, a možnosti tvorby rozhraní (interfaces). Uspořádanost datových toků a řídících procesů nám zajistí architektura MVC.

3.1.1 Návrh řešení MVC

Nároky na přehlednost, dlouhodobou udržitelnost a modularitu aplikace splňuje právě zmiňovaná architektura MVC. Dosahuje toho tím, že striktně odděluje prezentační, aplikační a doménovou logiku v systému. Rozdělení aplikace na moduly model, view a controller nám určuje odpovědnost jednotlivých zdrojových kódů. Vztahy mezi komponentami zase přesně vymezují procesy, které se odehrávají během zpracování uživatelova požadavku a odezvy na tento požadavek.

Jazyk PHP nenabízí žádné vestavěné prostředky (funkce nebo třídy), které by usnadňovaly tvorbu aplikace postavené na MVC. Existují však volně dostupné frameworky třetích stran, které architekturu MVC podporují. Tyto frameworky, mezi které lze zařadit Zend Framework nebo Nette Frameworky (od českého vývojáře), nabízejí hotové plně funkční knihovny (třídy a rozhraní) pro snadnou tvorbu aplikace založenou na MVC. Řešení třetích stran umožňují vývojářům webových aplikací oprostit se od samotné struktury aplikace a plně se věnovat její hlavní logice a podobě výstupu. Dva výše zmíněné frameworky nabízejí k použití dostatečně kvalitní realizaci MVC, která dokáže vyhovět většině nově vznikajících projektů. Pro potřeby této bakalářské práce a pro bližší pochopení filozofie MVC jsem se pokusil vytvořit vlastní řešení – „framework“.

Při návrhu vlastního řešení struktury aplikace jsem vycházel z obecného pojetí architektury MVC, přičemž v něm kombinuji některé vlastnosti variace MVC i MVP. Jelikož jazyk PHP neposkytuje komponentovou technologii pro tvorbu aplikace, jsou požadavky uživatele v mém návrhu zpracovávány controllerem a ne pohledem, jako je to v případě MVP. Tato vlastnost je tedy přebrána z varianty MVC. Inspirace architekturou MVP byla v případě odebrání vazby mezi modelem a komponentou view. Zrušení komunikace mezi těmito moduly vychází z návrhu *Passive view*, kterému jsem se věnoval na konci kapitoly 2.2.2 Varianta Model-View-Presenter. Pohled je proto plněn daty pomocí controlleru, který získává příslušná data z modelu. Kromě vkládání dat do view má controller také na starosti výběr příslušného pohledu. Následující obrázek názorně zobrazuje návrh vlastního řešení, které bylo v krátkosti v tomto odstavci popsáno. V ilustraci je také vidět interakce uživatele s aplikací a vztahy mezi jednotlivými komponentami.



Obrázek 6 – Návrh vlastního řešení MVC

3.1.2 Činnost a specifika webového controlleru

Controller je komponenta architektury MVC, která zpracovává požadavky uživatele. V prostředí webu jsou požadavky formulovány prostřednictvím URL adresy. Kromě jednoznačné identifikace požadovaného souboru, může Uniform Resource Locator (URL) obsahovat také parametry v podobě `&název=hodnota` (`&` – oddělovač parametrů). Parametry se v rámci URL uvádějí na konci. Podle názvu žádaného souboru nebo názvu a hodnoty parametru lze určovat controllery, které mají přebrat řízení aplikace. Příklady volání určitého controlleru:

- `http://www.domain.com/controller_directory/controller_name.php`
- nebo `http://www.domain.com/index.php?controller=name`

V prvním případě je spuštěn skript obsažený v souboru `controller_name.php`. Tento skript může obsahovat kód, který zajistí předání dat a zavolání potřebných metod modelu a pomocí patřičného view vygeneruje (X)HTML dokument, který je odeslán jako odpověď na požadavek uživatele.

Ve druhém příkladu je situace složitější. Informace o tom, který řadič má převzít odpovědnost za chod aplikace je předávána prostřednictvím parametru. Tento případ je charakteristický pro dnešní webové aplikace. Spočívá v tom, že přístup do aplikace je možný pouze přes jediný soubor typicky `index.php` nebo `default.php`. Toto řešení má značná pozitiva. Například je možné v rámci tohoto souboru vyřešit autentifikační a autorizační politiku aplikace, jednotně zajistit připojení k databázi nebo provádět logování. Avšak pro koncept MVC má tento přístup pro tvorbu aplikací jednu nevýhodu. Je nutné vytvořit nějaký mechanismus, který by z parametrů URL adresy získal konkrétní název controlleru a předal mu řízení.

Pro doplnění dodávám, že k parametrům uvedených v adrese URL lze v prostředí jazyka PHP přistupovat pomocí vestavěné (globální) proměnné `$_GET`. Tato proměnná je typu asociativního pole, ve kterém jsou uloženy veškeré parametry obsažené v URL. Index prvku tohoto pole představuje název parametru, hodnota prvku určuje hodnotu parametru.

Mechanismus, který zajišťuje delegování odpovědností konkrétním controllerům je tzv. *front controller* [7]. Front controller je ve skutečnosti třída, jejíž hlavním úkolem je analyzovat URL řetězec a na jeho základě vytvořit instanci takzvaného *action controlleru,* které mu je následně předáno řízení. Akční controller je opět třída obsahující metody, které spolu nějakým způsobem souvisejí, věnují se tedy řízení jedné společné oblasti v rámci doménové logiky. Metody akčních controllerů se nazývají akční metody neboli akce. Pro definování požadavku na konkrétní akční metodu se využívají také parametry URL adresy. Pro pochopení uvádím zjednodušený praktický příklad:

1. Uživatel kliknul v rámci webové stránky (administrační části redakčního systému) na tento dokaz:
   `http://www.domain.com/index.php?controller=article&method=delete&id=10`

2. Požadavek je odeslán na server a zpracován skriptem `index.php`, který vytváří instanci *front controlleru.*

3. Front controller získá z URL adresy tři parametry a jejich hodnoty:
   název akčního controlleru: `article`,
   název akční metody: `delete`,
   identifikátor článku: `10`.

4. Front controller vytvoří instanci article controlleru a zavolá jeho metodu `delete` s parametrem `10`:
   ```
   <?php
        $articleController = new ArticleController();
        $articleController->delete(10);
   ?>
   ```

5. Metoda `delete` zajistí provedení potřebných operací na modelu, na základě výsledku operace zvolí patřičný pohled, naplní ho daty (pokud je to nutné) a odešle zpět uživateli.

Tento mechanismus delegování odpovědnosti řízení konkrétním controllerům na základě uživatelského požadavku využívá i moje mírně složitější řešení. Popis vlastních tříd a rozhraní spadajících v rámci architektury MVC do komponenty controller:

Třída FrontController

Jak již bylo řečeno třída `FrontController` získá názvy a hodnoty parametrů předávané pomocí adresy URL. Tyto parametry vyhodnotí a na jejich základě zavolá příslušnou metodu akčního controlleru.

Pro odlišení tříd v rámci aplikace se názvy akčních controllerů skládají ze dvou částí: oblast zájmu controlleru a samotného slova „Controller“. Příklad pojmenování třídy controlleru, který řídí činnosti vykonávané nad daným článkem: `ArticleController`. V parametru URL je předávána pouze první část názvu controlleru. Slovíčko „Controller“ je k názvu připojeno v těle metody `FrontControlleru`. Před názvy akčních metod jednotlivých controllerů je analogicky přidáno slovíčko „action“.

Při analýze řetězce a delegování řízení mohou nastat tyto případy:

a) Adresa URL obsahuje hodnoty parametrů `ctrl` (název *action controlleru*) a `do` (název akční metody). Pokud existuje třída s názvem uvedeným v parametru `ctrl` a tato třída implementuje i požadovanou akční metodu, je vytvořena její instance a daná metoda zavolána. Pokud třída nebo metoda neexistuje je řízení předáno takzvanému *error controlleru* (viz popis třídy `ErrorController`).

b) Adresa URL obsahuje pouze název controlleru. Pokud daný controller existuje, je opět vytvořena jeho instance a spuštěna metoda s názvem `actionDefault`, kterou musí každý akční controller povinně implementovat.

c) Adresa URL neobsahuje parametr `ctrl` ani `do`. Potom je vytvořena instance třídy `DefaultController` a vyvolána metoda `actionDefault`. Speciální třída `DefaultController` musí být součástí každé nově vytvořené aplikace postavené na tomto popisovaném frameworku.

Třída implementuje rozhraní `IFrontController` a obsahuje tyto metody:

- `delegate` – veřejná metoda, jejíž náplní je předání řízení akčnímu controlleru,
- `setActionController` a `setActionMethod` – privátní metody, které analyzují parametry `ctrl` a `do` obsažené v adrese URL.

Zdrojový kód třídy `FrontController` je uveden v příloze.

Rozhraní IFrontController

Toto rozhraní implementuje třída `FrontController`, která musí zahrnout metodu `delegate`.

Rozhraní IActionController

Rozhraní `IActionController` implementují jednotlivé akční controllery. Chování těchto řadičů je závislé na konkrétních vlastnostech aplikace. Jedinou povinností specifických controllerů je implementovat metodu `actionDefault`.

Abstraktní třída AActionController

Tato třída je abstraktním předkem pro akční controllery. Obsahuje chráněný atribut `view`, ke kterému mají přímý přístup pouze potomci této třídy. Atribut `view` je instancí třídy `View`, která je popsána v níže uvedené kapitole o vlastním řešení komponenty view. Tato vlastnost umožňuje každému akčnímu controlleru zvolit požadovaný pohled, naplnit ho daty a odeslat uživateli.

Abstraktní třída dále obsahuje abstraktní metodu `redirect`, která dovoluje controllerům přesměrovat na jiný akční controller nebo jinou akční metodu.

Třída DefaultController a ErrorController

Tyto dvě třídy nejsou součástí hlavní knihovny (jádra) aplikace, ale jejich existence je nezbytná pro korektní chod systému. Jejich činnost může být pro různá nasazení tohoto frameworku odlišná. Záleží tedy na autorovi aplikace, jaké chování těmto třídám nadefinuje.

Třída `DefaultController`, jak už její název napovídá, funguje jako základní controller. Je použit v případě, kdy není prostřednictvím URL předána informace o názvu controlleru. Například:

- `http://www.domain.com/`,
- `http://www.domain.com/index.php`
- nebo `http://www.domain.com/index.php?ctrl=`.

Instance `ErrorControlleru` je vytvořena, jakmile `FrontController` zjistí, že akční controller nebo metoda zadaná v adrese URL neexistuje. `ErrorController` může být ale také spuštěn explicitně:

a) buď vytvořením odkazu v pohledu:
   `http://www.domain.com/index.php?ctrl=Error`

b) nebo zavoláním metody `redirect`, kterou každý akční controller zdědí z abstraktní třídy `AActionController`:
   `$this->redirect('Error');`.

Závěr této podkapitoly, která popisovala vlastní řešení činnosti řadiče v architektuře MVC, doplním o vysvětlení, jak snáze dosáhnout sloučení stejného popřípadě podobného chování jednotlivých akčních controllerů a zamezit tak duplicitám v kódu. Toto je možné zajistit vytvořením abstraktního controlleru, který v sobě implementuje shodné chování akčních controllerů. Akční controllery, využívající tuto společnou funkcionalitu, se stanou potomky vytvořeného abstraktního controlleru a potřebné metody zdědí. Příkladem společné „práce“ může být jednotný způsob generování hlavičky nebo získávání dat pro menu v rámci webové stránky. Ve společném abstraktním controlleru je možné také jednoduše vyřešit připojení k databázi nebo autorizovaní uživatelů. Deklarace takto zhotovených tříd abstraktních controllerů by měly mít příznak `abstract`. To zamezí vytvoření instance této třídy. Abstraktní controllery by totiž neměly obsahovat akční metody s řídící logikou, ale pouze společné části kódů určené pro akční řadiče.

3.1.3 Řešení komponenty view

V kapitole 3.1.1 Návrh řešení MVC jsem se v krátkosti zmínil, jak část view v rámci vlastního řešení architektury MVC funguje. Její vlastnosti vychází z konceptu *Passive view*, nemá tedy žádnou vazbu na model a prakticky o modelu „nic neví“. Za předání dat pohledu i jeho výběr je odpovědný controller, který potřebná data získá z modelu. Každý akční controller obsahuje chráněný atribut `view`, který zdědí od abstraktní třídy `AActionController`. Tento atribut je instancí třídy `View`. Pomocí metod této třídy je možné zvolit patřičný pohled, odevzdat mu data a přikázat mu, aby se vykreslil, tzn., odeslal výstup aplikace uživateli.

Moje řešení modulu view je tedy realizováno pomocí tří tříd `View`, `Template` a `Layout`. Následuje popis jejich vlastností a chování.

Třída View

Třída `View` je hlavní třídou tohoto modulu. Vlastní dva atributy a nabízí pět metod. Nejdůležitější veřejný statický atribut `templates` je typu pole. Slouží k uchování šablon respektive instancí třídy `Template`. Atribut je veřejný a statický z toho důvodu, že je k němu přistupováno z jednotlivých objektů, které představují samostatné šablony. Druhým, privátním atributem je `layout`, v němž je uložen objekt typu `Layout`. Metody třídy:

- Metoda `setLayout($file, array $data)` nastavuje šablonu (soubor s příponou phtml), sloužící jako layout stránky, a data pro ni určená.
- Metoda `addLayoutData(array $data)` umožňuje dodatečně přidat další data do šablony layout.
- Metoda `addTemplate($templateName, $file, array $data)` vytvoří z argumentů metody objekt typu `Template` a přidá ho do atributu

`templates`. Argument `templateName` představuje jednoznačný identifikátor šablony, který je potřebný při kombinování šablon (viz příklad níže na kombinování šablon).

- Metoda `addTemplateData($templateName, array $data)` je analogická metoda k druhé popisované metodě.
- Metoda `render()` se stará o „includování“ šablon. Poskládá šablony do pořadí, aby vznikl souvislý kód tvořící výslednou webovou stránku.

Třída Template

Třída `Template` představuje šablonu. Zapouzdřuje název souboru, ve kterém je kód šablony, a data určená pro ni. Obsahuje také metodu `paste()`, která slouží k „vykreslení“ šablony (pomocí řídí struktury PHP `include`).

Třída Layout

Tato třída je potomkem třídy `Template`. Liší se pouze definicí metody `paste()`, která vychází z toho, že soubory představující layout stránky se nacházejí v jiné složce než běžné šablony. Toto opatření existuje z důvodu přehlednosti.

Mimo výše zmíněných tří tříd patří do komponenty view také soubory s koncovkou pthml. Jedná se o takzvané šablony, jejichž úkolem je generování kódu ve formátu XHTML. Tyto šablony obsahují elementy (tagy) značkovacího jazyka XHTML a základní funkce a řídící příkazy jazyka PHP (`if-else`, `for`, `foreach`, `echo`, `isset`, `empty`). V rámci šablony by neměla být uvedena žádná složitá rozhodovací logika a také by neměla existovat žádná závislost na metodách a třídách modelu popřípadě controlleru.

Řešení je navrženo tak, aby bylo možné kombinovat libovolné šablony dohromady a také je i použít opakovaně v rámci jednoho pohledu. Viz následující ukázka:

- Kombinování šablon resp. jak jsou vkládány šablony do sebe:

```
<body>
    <?php View::$templates['page']->paste(); ?>
</body>
```

  Tento kód vkládá mezi elementy `body` šablonu s jednoznačným názvem „`page`“.

- Přístup šablony k jejím datům. Ukázka části šablony (article.phtml), která má za úkol zobrazit článek:

```
<?php
     $article = $this->data['article'];
?>

<div class="article">
     <h2><?php echo $article['title']; ?></h2>
     ...
</div>
```

Tento kód získá z dat určených pro tuto šablonu hodnotu proměnné `article` a vloží ji do elementu `h2`.

Kapitolu o komponentě view ukončím krátký příkladem toho, jak jednoduše v rámci akčního controlleru lze vytvořit pohled, předat mu data a vykreslit ho.

```
...
$mainData['article']['title'] = 'Nadpis článku';
$mainData['article']['content'] = 'Obsah článku';
$this->view->addTemplate('main', 'article.phtml', $mainData);

$layoutData['pageTitle'] = 'Titulek stránky';
$this->view->setLayout('general.phtml', $layoutData);

$this->view->render();
```

3.1.4 Jednoduchost návrhu modelu

Komponenta model je už z principu MVC nezávislá na typu aplikace. Kromě dat obsahuje model jednotlivé třídy nebo funkce řešící hlavní (business) logiku aplikace. Logika modelu by měla být využitelná jak na webových tak desktopových (okenních či konzolových) aplikacích. Model tedy nepodléhá struktuře controlleru ani pohledu.

Následující ukázka demonstruje přístup *action controlleru* k modelu. Zdrojový kód obsahuje úryvky akční metody controlleru, který řídí činnost při zpracování požadavku uživatele na smazání článku:

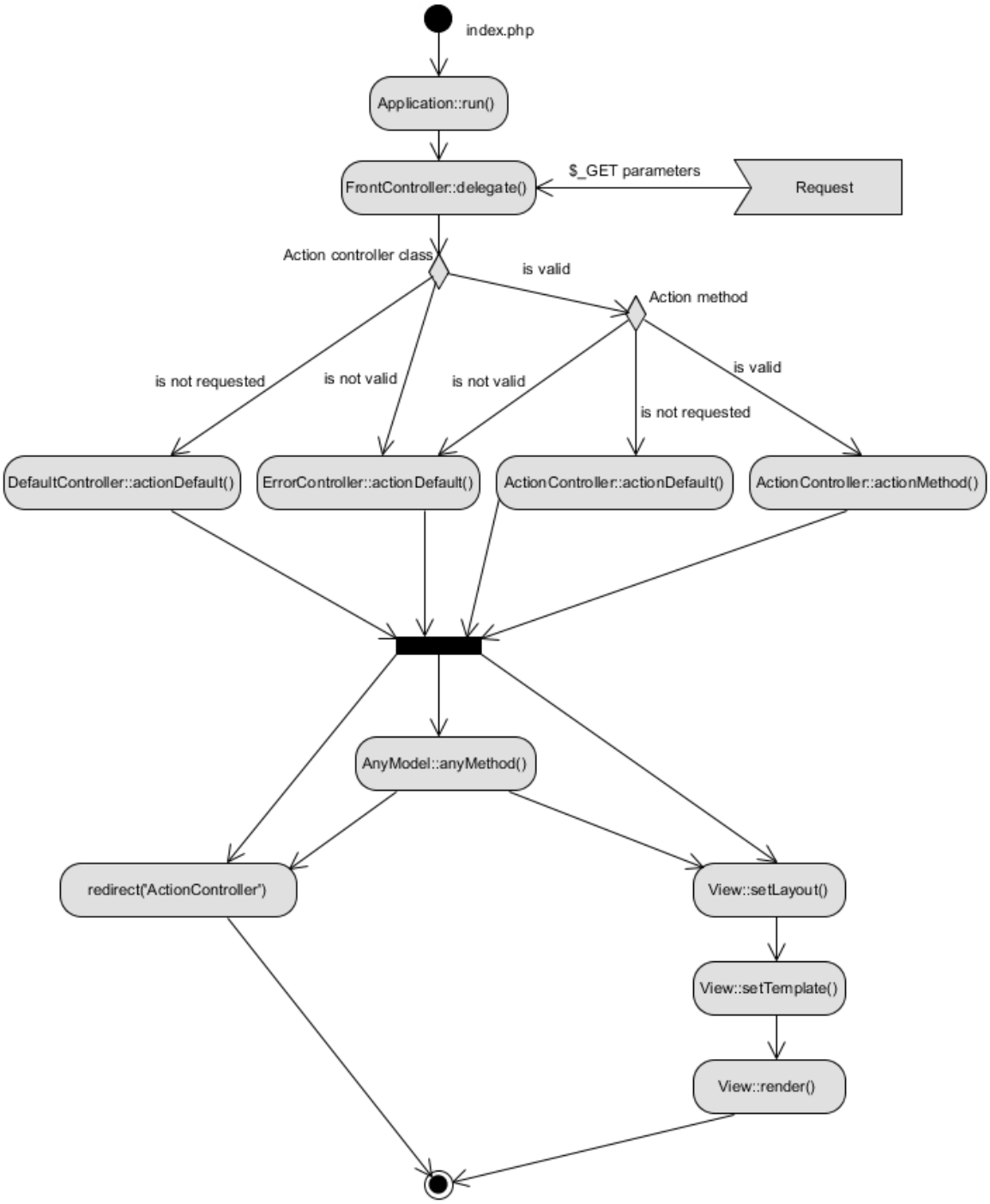
```
...
$cms = new CMS();

if (!$cms->isAuthorized(Action::DELETE, Object::ARTICLE,
$article)) {
     $this->redirect('SignOn');
}

if ($cms->deleteArticle($article)) {
     $this->redirect('BOArticles');
} else {
     ...
}
...
```

První řádek vytváří objekt podle předpisu třídy `CMS` (Content Management System), která je součástí modelu. Tato třída nabízí mimo jiné metodu na ověření uživatele a vymazání určeného článku (viz podkapitola Třída CMS).

Příklady tříd, které mají různé povinnosti v rámci aplikace, spadajících do komponenty model:

- Třída `ACL` (Access Control List) obsahuje logiku mající na starosti autorizaci uživatelů.
- Třída `ContactEditForm` zahrnuje metody obsluhující formulář pro editaci kontaktu. Hlavní metody této třídy například ověřují:
    - data předávaná formulářem,
    - zda byl formulář odeslán (stisk tlačítka submit)
    - nebo zda byly změněny výchozí hodnoty formuláře.
- Třída `Article` představuje článek z reálného světa. Především tedy zapouzdřuje data patřící článku (nadpis, obsah, čas vytvoření, stav, autora atd.).
- Třída `UserTable` a třídy jí podobné představují rozhraní pro přístup k tabulkám uložených v databázi (User). Metody této třídy vykonávají SQL dotazy nad touto tabulkou.
- Třída `CMS` je hlavní doménovou třídou celého redakčního systému. Metody této třídy tvoří jednotné rozhraní k modulům pro správu článků, uživatelů a kontaktů.

Pro názornost zde uvádím activity diagram, který popisuje chování uvnitř (v jádru) webové aplikace postavené na vlastním řešení architektury MVC. Diagram znázorňuje procesy (akce), které mohou nastat při zpracování uživatelského požadavku.



Obrázek 7 – Activity diagram zpracování požadavku v MVC

3.2 Komplexní řešení aplikace

Po vysvětlení realizace softwarové architektury MVC uvedené v předchozí kapitole se druhá část praktické části bakalářské práce věnuje řešení problematiky týkající se konkrétního chodu a funkcionalitě portálu SK Studenec.

Shrňme pro začátek konkrétní a obecné požadavky na fungování portálu. Přání představitelů oddílu SK Studenec na funkčnost portálu byly tyto:

- rozdělení portálu na sekce, které se zabývají odlišnou sportovní aktivitou (kopaná, orientační běh, alpské lyžování),
- rozdělení fotbalové sekce na týmy (muži A, muži B, dorost, starší žáci, mladší přípravka, internacionálové),
- vytvoření administrační části (back-end aplikace) s možností zabezpečeného přístupu,
- vytvoření uživatelských rolí (administrátor, korektor, redaktor), která mají odlišná oprávnění,
- vytvoření modulu pro správu článků (vytvoření, editace, mazání, publikování článků) s možností přiřadit článek k libovolné sekci resp. týmu a určit přesný datum a čas, kdy má být článek publikován,
- možnost generování článků ve formátu PDF,
- vytvoření modulu pro správu uživatelů (vytvoření, editace a rušení uživatelů), který mají možnost obsluhovat pouze administrátoři,
- vytvoření modulu pro správu kontaktů (přidávání, editace a odebírání kontaktů), které jsou generovány v podobě seznamu ve front-endu aplikace.

Obecné požadavky na tvorbu moderní, rozsáhlejší webové aplikace jsou:

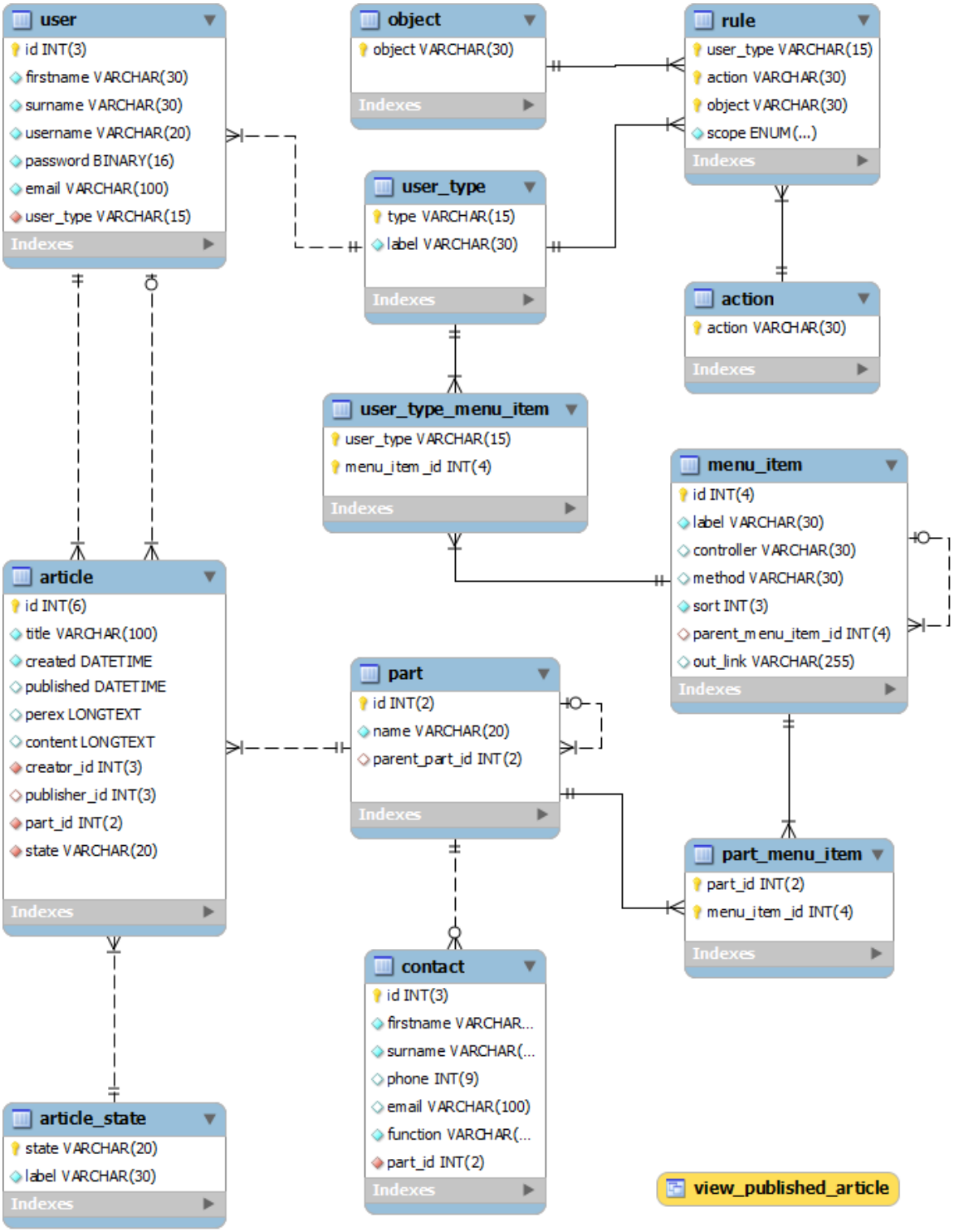
- moderní, přehledný a čistý design prezentační části aplikace,
- přehledná, na ovládání jednoduchá a intuitivní administrační část aplikace,
- eliminace generování chyb při běhu v reálném prostředí,
- vhodné řešení autentifikace a autorizace uživatelů,
- zamezení SQL injection,
- vhodný návrh struktury databáze,
- validita stránek,
- dostatečná SEO optimalizace.

V následujících podkapitolách se zaměřuji na konkrétní řešení některých požadavků, které byly zmíněny výše.

3.2.1 Struktura databáze

Veškerá dynamicky zpracovávaná data uchovává tato webová aplikace v databázi. Tím je umožněna lepší manipulace s daty a souběžný přístup více uživatelů k těmto datům. Struktura databáze je postavená na relačním databázovém systému MySQL. Databáze je navržena tak, že obsahuje 12 tabulek, z nichž většina podléhá minimálně třetí normální formě. Součástí databáze je také jeden pohled.

ER Diagram



Obrázek 8 – ER diagram návrhu databáze

Popis databázových objektů

Tabulka user uchovává záznamy o všech registrovaných uživatelích. Tito uživatelé mají přístup do administrační části portálu SK Studenec. Hodnoty ve sloupci `password` jsou z důvodu bezpečnosti šifrovány pomocí funkce `password` resp. `old_password` (vestavěné funkce systému MySQL). Uživatelské heslo není proto možné zpětně určit. Uživatelská role účastníka systému je dána atributem `user_type`. Tato tabulka podléhá 3. normální formě.

Tabulka user_type je výpisem všech možný uživatelských rolí. Obsahuje tři hodnoty: *admin*, *corrector* a *redactor*. Tabulka `user_type` splňuje požadavky pro pátou normální formu. Uživatelským rolím a jejich oprávněním se věnuji v samostatné kapitole.

Tabulka article zaznamenává články v redakčním systému. Pro role administrátora a korektora jsou přístupné všechny články. Redaktor vidí pouze vlastní a publikované články. Pomocí referenční integrity je v tabulce uložena informace o stavu článku, sekci, do které spadá, a uživateli, který článek vytvořil a publikoval. Tabulka `article` odpovídá 3. normální formě.

Tabulka article_state je výčet stavů, kterých může článek nabývat. Stav článku určuje jeho fázi vývoje a možnost přístupu uživatelů k němu. V redakčním systému existují tři stavy článku:

- *Draft (koncept)* – označení článku, který je rozpracovaný. Není tedy určený k publikování. Pro uživatelskou roli redaktora jsou přístupné pouze vlastní rozpracované články (ve stavu koncept). Administrátor i korektor vidí drafty všech redaktorů.

- *Approval process (čekající na schválení)* – stav článku určený především pro redaktory. Redaktor nemá možnost převést článek přímo do stavu publikovaný. Označení článku tímto stavem dává redaktor signál korektorům či administrátorům, aby článek zkontrolovali a nastavili mu stav publikovaný.

- *Published (publikovaný)* – označuje článek jako publikovatelný. Tento stav je nutnou ale ne postačující podmínkou k tomu, aby byl článek přístupný běžnému návštěvníkovi stránek. Druhým nutným a posledním předpokladem je čas publikování, jenž je možné každému článku nastavit. Pouze čas, který je starší než aktuální, umožňuje článek uveřejnit.

Tabulka `article_state` splňuje pátou normální formu.

Tabulka contact obsahuje záznamy týkající se správy kontaktů v rámci redakčního systému. Tabulka kontaktů plní třetí normální formu v rámci normalizace databáze.

Tabulka part vystihuje hierarchickou strukturu oddílu SK Studenec. Sloupeček `parent_part_id` odkazuje na nadřazenou sekci ve stromové struktuře klubu, jejíž záznam se nachází v této tabulce. Jde tedy o rekurzivní vztah v relačním databázovém modelu. Tato tabulka splňuje třetí normální formu. Nástin aktuálního hierarchického složení oddílu:

```
SK Studenec (1. úroveň)
     Kopaná (2. úroveň)
          Muži A (3. úroveň)
          Muži B
          Dorost
          Starší žáci
          Mladší přípravka
          Internacionálové
     Orientační běh
     Alpské lyžování
```

Tabulka rule společně s tabulkami `action` a `object` pomáhá řešit autorizační politiku systému. Tabulka obsahuje záznamy, které jsou kombinací čtyř hodnot: uživatelská role, akce, objekt a rozsah. Hodnoty ve sloupečku `user_type` mohou nabývat hodnot z tabulky `user_type`. Hodnoty ve sloupci `action` říkají, jaká činnost je daným uživatelem požadována. Atribut `object` uchovává hodnoty, které vyjadřují entitu, nad níž je daná akce vykonávána. Poslední sloupec `scope` je výčtem tří hodnot: `all`, `own` a `none`. Určuje rozsah působnosti. Požadovaná akce může tedy být vykonávána nad všemi objekty daného typu nebo pouze nad daným uživatelem vlastněnými objekty nebo nad žádnými. Tabulka `rule` odpovídá třetí normální formě.

Tabulka action obsahuje položky, které vyjadřují akce, jenž je možné provádět nad danými objekty. Doposud jsou v ní uvedeny tyto hodnoty: `create`, `delete`, `list`, `publish`, `read`, `edit`. Tabulka patří do 5. normální formy.

Tabulka object doplňuje tabulku `action`. Zahrnuje objekty, s nimiž uživatel manipuluje a zároveň podléhají autorizační politice. Objekty, nad kterými je možné provádět výše zmíněné akce: `article`, `contact` a `user`. Tabulka obsahuje pouze jeden atribut, proto také spadá do páté normální formy.

Tabulky menu_item a následujících dvou tabulek využívá aplikace k sestavení menu jak ve front-endové části tak back-endové. Tabulka obsahuje jednotlivé položky menu. Záznam položky obsahuje název a odkaz (URL adresu) popřípadě controller a jeho akční metodu. Tato tabulka splňuje pouze druhou normální formu a to z důvodu, že atribut `method` je závislý na atributu `controller`. Tento nedostatek by se dal vyřešit

dekompozicí tabulky `menu_item`, čímž by z atributů `method` i `controller` vznikly dvě nové tabulky. Dále by bylo nutné vytvořit dvě asociativní tabulky řešící vztah více k více. Jelikož počet položek tabulky `menu_item` není nijak závratný, manipulace se záznamy probíhá zřídka, není nutné vyžadovat, aby tabulka splňovala třetí normální formu.

Tabulka user_type_menu_item a part_menu_item jsou asociativní tabulky řešící vztah M:N mezi tabulkami `user_type` a `menu_item`em respektive `part` a `menu_item`em. Obě tyto tabulky splňují pátou normální formu při řešení normalizace databázového modelu.

Pohled view_published_article extrahuje z tabulky `article` články, které splňují podmínky publikování. Záznamy v tomto pohledu musí obsahovat ve sloupečku `state` hodnotu `published`. Další podmínkou je, že datum publikování (atribut `published`) musí být starší než datum aktuální. S tímto pohledem je pracováno v rámci front-endu aplikace.

Databázový systém MySQL, který má na starosti uchovávání dat pro portál SK Studenec, využívá formát úložiště dat (*storage engine*) InnoDB. Tento typ úložiště umožňuje v rámci relační databáze využít výhody referenční integrity, která zajišťuje zadávání správných hodnot do sloupců, které jsou cizími klíči. Další výhodou InnoDB je možnost zpracování transakcí, čehož je také při obsluze databáze využito. Pro úplnost dodávám, že v základním nastavení databázového systému MySQL se používá formát úložiště dat MyISAM, které výhody InnoDB nenabízí.

Sloupce označené `id` v tabulkách `user`, `article` a `contact` mají aktivní atribut `AUTO_INCREMENT`, který zajišťuje generování hodnot celých čísel pro jednoznačnou identifikaci záznamu v tabulce.

3.2.2 Adresářová struktura aplikace

Adresářová struktury aplikace je koncipována tak, aby promítala architekturu MVC. Jednotlivé zdrojové kódy (definice tříd) jsou rozděleny do složek podle komponent model, view a controller. Použitá adresářová struktura také odděluje zdrojové kódy týkající se pouze konkrétní aplikace a takzvané knihovny. Knihovny jsou soubory použitelné bez úprav v různých typech aplikací. Tato struktura zlepšuje orientaci v množství tříd, které jsou součástí celého systému.

Z důvodu bezpečnosti je přístup do jednotlivých složek řízen pomocí souborů `.htaccess`. Tento mechanismus nabízí server Apache. Konfigurací těchto souborů lze z pohledu klienta různě omezit práci se soubory uvnitř složek.

```
/ (document root)
     app/*
          controllers/*
               abstract/
          models/*
               acl/
               forms/
               modules/
          views/*
               layouts/*
               templates/*
     libs/*
     public/*
          images/*
          scripts/*
          styles/*
     index.php
```

Hierarchii a názvy adresářů označených hvězdičkou je nutný dodržet pro korektní chod aplikace. Následuje popis důležitých složek.

- `app/` – třídy vztahující se k chování specifické aplikace,
- `controllers/` – akční controllery sloužící k řízení aplikace a obsloužení požadavků uživatele.
- `abstract/` – abstraktní controllery slučující společné chování akčních controllerů,
- `models/` – třídy řešící doménovou logiku aplikace,
- `acl/` – třídy zabezpečující autorizaci uživatelů,
- `forms/` – třídy pro obsluhu webových formulářů,
- `modules/` – složky s třídami řešící problematiku jednotlivých modulů redakčního systému (moduly: users, articles, contacts, parts, menus),
- `views/` – šablony (soubory s koncovkou phtml), obsahující převážně XHTML kód, pro generování výstupu v podobě webové stránky,
- `layouts/` – šablony rozvržení stránky (obsahují například hlavičku XHTML a elementy `head` nebo `body`),
- `templates/` – šablony generující XHTML kód různých částí webové stránky,
- `libs/` – knihovny PHP kódu (třídy, funkce) řešící obecnou problematiku webových aplikací, která není závislá na konkrétním použití, a knihovny od jiných tvůrců (architektura MVC – jádro, databázový přístup, výjimky, autentifikace, datové struktury),

- `public/` – veřejně dostupné dokumenty,
- `images/` – rastrová grafika určená pro vzhled stránek,
- `scripts/` – klientské skripty (javascriptové soubory),
- `styles/` – kaskádové styly (CSS soubory).

3.2.3 Uživatelské role

Uživatele systému je možné rozdělit do dvou kategorií. Do první kategorie spadá běžný návštěvník (host, guest) portálu SK Studenec. Ten má přístup pouze k prezentační části webové aplikace (front-endu). Může prohlížet zveřejněný obsah stránek, ale nemá žádnou možnost tento obsah vytvářet nebo měnit. Návštěvník portálu má tedy zatím možnost prohlížet publikované články redakčního systému a kontakty.

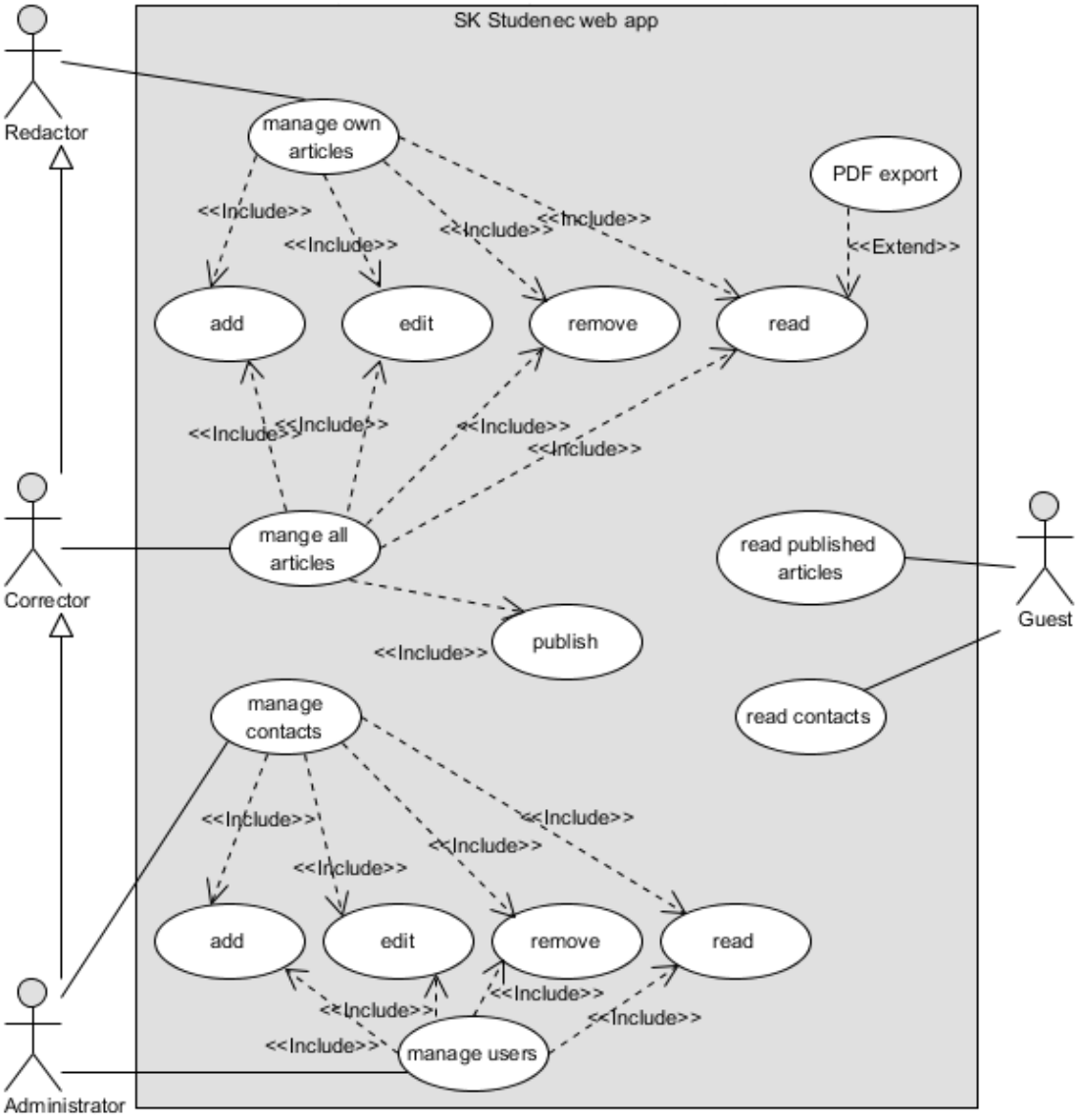
Druhá kategorie zahrnuje uživatele, kteří mají v rámci systému přidělenu konkrétní uživatelskou roli. Tito uživatelé mají přístup do administrační části redakčního systému (back-end), kde mají možnost tvořit obsah portálu. V rámci systému existují tři uživatelské role, které se liší různým rozsahem oprávnění.

- **Redaktor** je uživatelská role s nejmenšími právy. Pro tento typ je v administrační části aplikace zpřístupněn pouze modul pro správu článků. Redaktoři mohou vytvářet, editovat a mazat vlastní články, ale nemohou články publikovat. Možnost zveřejnění článku mají až korektoři nebo administrátoři. Redaktor může článku přiřadit stav „čeká na schválení“ a tím dát pověřeným uživatelským rolím signál, že daný článek je dokončený a připravený k publikaci. Role s vyšším oprávněním článek zkontrolují a nastaví mu stav „publikovaný“, čímž je uveřejněn v prezentační části aplikace.

- **Korektor** rozšiřuje pravomoci redaktora. Korektoři mají také možnost spravovat pouze články v rámci redakčního systému. Oproti redaktorům mohou manipulovat nejen s vlastními články, ale i s články ostatních uživatelů. Tato uživatelská role může také články uveřejňovat.

- **Administrátor** je hlavní a nejdůležitější uživatelskou rolí. Kromě modulu se správou článků je administrátorům také zpřístupněn modul pro správu uživatelů a kontaktů. Možnosti práce s články má tato role stejné s korektory. Funkce, které má administrátor k dispozici v rámci modulů pro správu uživatelů a kontaktů shrnuje *Use case diagram* na následující stránce. Počet administrátorů v systému není omezen.

Návštěvník stránek, který má zájem podílet se na tvorbě obsahu, musí pro vytvoření uživatelského účtu a přidělení role požádat administrátora systému.



Obrázek 9 – Use case diagram portálu SK Studenec

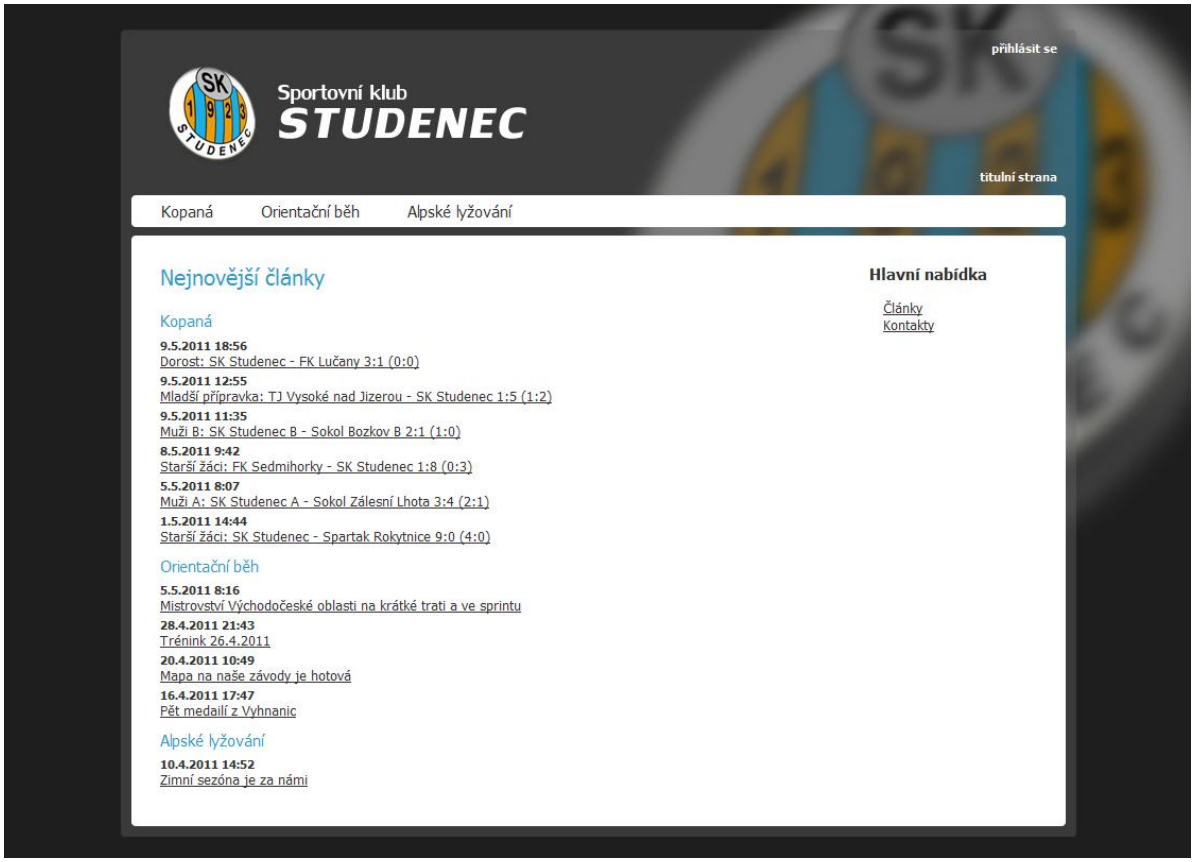
3.2.4 Rozvržení a design stránek

Při řešení vzhledu a rozvržení stránek pro portál SK Studenec jsem se snažil vyhovět požadavkům na jednoduchost a přehlednost a to jak v prezentační tak v administrační části aplikace. Zároveň jsem se pokusil vytvořit stránky, které budou moderní a uživatelsky přitažlivé.

Samozřejmostí při tvorbě webové aplikace je validita stránek. Pro popis struktury jednotlivých dokumentů (stránek), které jsou vráceny uživateli jako reakce na jeho požadavek, je využita specifikace *XHTML 1.0 Transitional*. Je nutné podotknout, že obsah portálu SK Studenec je tvořen registrovanými uživateli, kteří nemusejí mít představu o validaci. Jimi vytvořené články mohou zahrnovat některé prvky, díky nimž se stránky

stanou nevalidní. Na tuto možnost je však myšleno a většina nebezpečných prvků je před uložením do databáze filtrována.

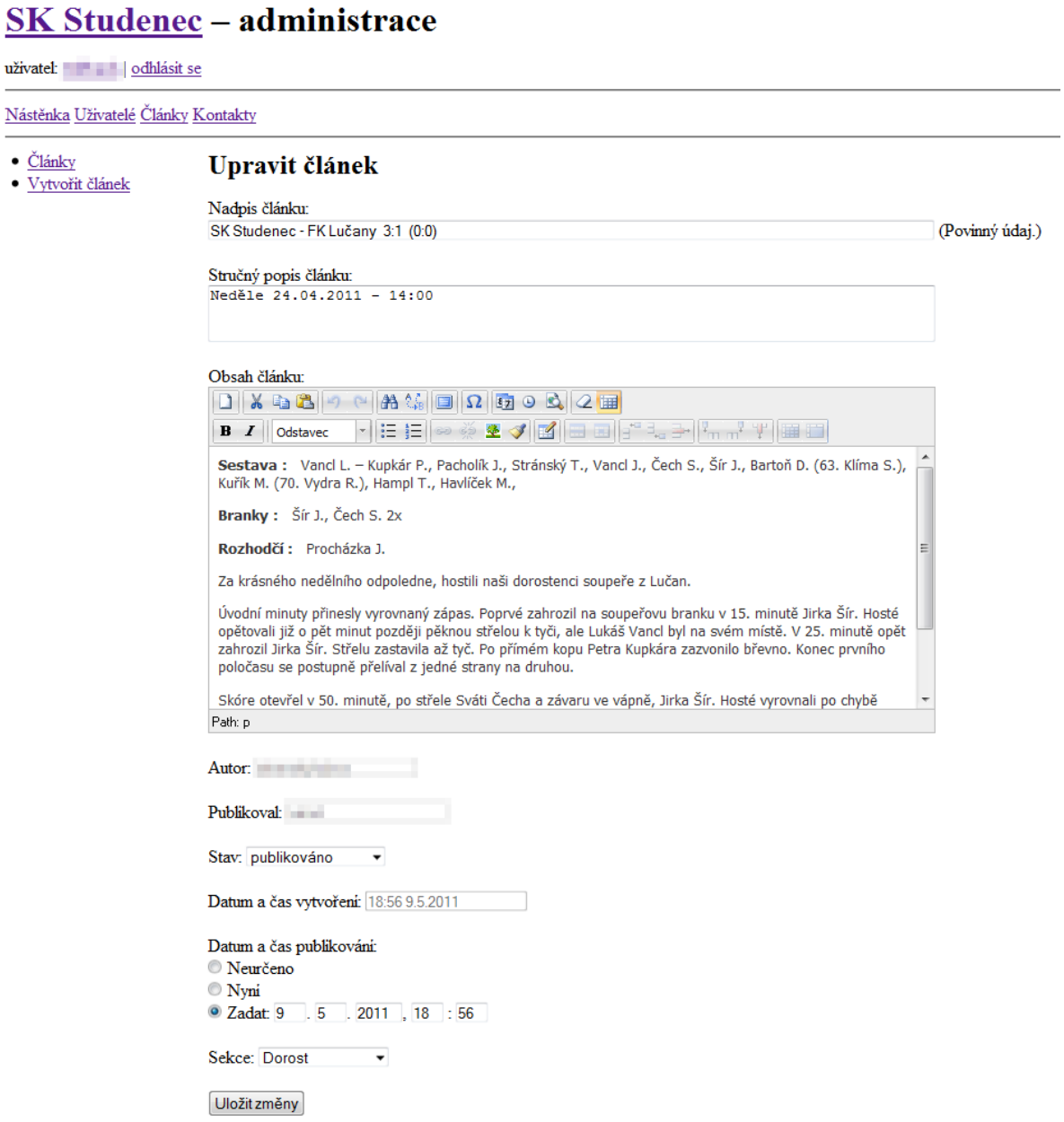
Prezentační část (front-end) portálu SK Studenec vychází z klasického layoutu rozvržení stránky. Vrchol stránky patří hlavičce, která se skládá z loga a názvu oddílu. Pod hlavičkou následuje horizontální menu, které nabízí vstup do jednotlivých sekcí oddílu. Pokud je sekce rozdělena na skupiny (viz kopaná), jsou tyto skupiny zobrazeny jako druhá úroveň tohoto menu. Prostor pod hlavičkou a horizontální nabídkou je rozdělen na dvě části uspořádané vedle sebe. Pravá, užší část je určena pro hlavní nabídku (vertikální), která zahrnuje odkazy na články a kontakty dané sekce. Levá, širší část je vymezena pro hlavní obsah stránky (seznam článků, článek, seznam kontaktů, …). V pravé horní části stránky se nachází odkaz na formulář pro vstup do administrační části systému. Na titulní stránce portálu, jejíž náhled je níže k dispozici, je zobrazen aktuální seznam nejnovějších článků v jednotlivých sekcích.



Obrázek 10 – Prezentační část portálu SK Studenec

Administrační část (back-end) aplikace slouží ke správě redakčního systému. Tato část portálu si neklade za cíl zapůsobit svým vzhledem na uživatele. Naopak je prostá a intuitivně ovladatelná.

Hlavička kromě nadpisu zobrazuje uživatelské jméno aktuálně přihlášeného uživatele a nabízí odkaz na odhlášení se ze systému. Pod hlavičkou je horizontální nabídka všech modulů, ke kterým má daný uživatel přístup a které může využívat. Kromě toho obsahuje tato nabídka položku „Nástěnka“, která odkazuje na výchozí stránku administrační části a zobrazuje přehled vlastní činnosti v systému. Levá, postranní nabídka nabízí možnosti a funkce konkrétního modulu. Zbylá a podstatná část stránky obsahuje různé ovládací prvky pro manipulaci s obsahem. Následují screenshot ukazuje například prostředí pro editaci článku.



Obrázek 11 – Administrační část portálu SK Studenec

4 Závěr

Cílem této bakalářské práce bylo vytvořit webový portál pro oddíl SK Studenec. Součástí portálu měl být redakční systém se správou uživatelů, článků a kontaktů. To vše jsem se snažil splnit s ohledem na požadavky, kterými byly jednoduchost, přehlednost a rozšiřitelnost systému.

V rámci teoretické části bakalářské práce jsem charakterizoval a zhodnotil architekturu pro vývoj softwaru Model-View-Controller, která mi ke splnění zmíněných požadavků pomohla. Poznatky ze studia této architektury i s jejími výhodami jsem se snažil uplatnit při realizaci portálu. Vlastní implementace MVC nebyla obtížná, i když hodně souvislostí ohledně principu architektury MVC mi docházelo až při samotném psaní kódu. O výhodách této architektury a správnosti vlastního řešení jsem se přesvědčil při dodatečném programování modulu pro správu kontaktů. Modularita systému byla tedy ověřena.

Portál SK Studenec je přístupný a plně funkční na adrese www.skstudenec.cz od konce března 2011. V systému během prvního měsíce provozu bylo zaregistrováno 13 uživatelů a vytvořeno 40 článků. Aplikace není ve finální podobě, možností na vylepšení a rozšíření je několik. Avšak díky architektuře MVC, na které je aplikace postavena, nebude obtížné změny nebo rozšíření realizovat.

Závěr bakalářské práce bych chtěl zakončit tvrzením, že vývoj webových aplikací je během na dlouhou trať a že nikdy nebudou vyčerpány možnosti, jak aplikaci zdokonalit.

Literatura

[1] **Baray, Cristobal. 1999.** The model-view-controller (MVC) design pattern. [Online] 1999. [Citace: 1. 5 2011.] http://cristobal.baray.com/indiana/projects/mvc.html.

[2] **Bernard, Borek. 2009.** Seriál MVC a další prezentační vzory. *Zdroják.cz.* [Online] 2009. [Citace: 1. 5 2011.] http://zdrojak.root.cz/serialy/mvc-a-dalsi-prezentacni-vzory/.

[3] **Lecky-Thomson, Ed a Nowicki, Steven D. 2010.** *Programujeme profesionálně.* Brno : Computer Press, 2010. str. 718. ISBN 978-80-251-3127-5.

[4] **Meyer, Eric A. 2007.** *Eric Meyer o CSS – Kompletní průvodce.* Brno : Zoner software, 2007. str. 560. ISBN 978-80-86815-64-0.

[5] **Potel, Mike. 1996.** MVP: Model-View-Presenter, The Taligent Programming Model for C++ and Java. *Wildcrest Associates.* [Online] 1996. [Citace: 1. 5 2011.] http://www.wildcrest.com/Potel/Portfolio/mvp.pdf.

[6] **Sevrjukov, Alexandr. 2007.** Co je Model View Controller? *PHP Model View Controller framework.* [Online] 2007. [Citace: 1. 5 2011.] http://pvc.boolean.cz/?section=basic&page=mvc.

[7] **Sun Microsystems. 2002.** Core J2EE Patterns – Front Controller. [Online] 2002. [Citace: 1. 5 2011.] http://java.sun.com/blueprints/corej2eepatterns/Patterns/FrontController.html.

[8] **—. 2002.** Designing Enterprise Applications with the J2EE Platform. [Online] 2002. [Citace: 1. 5 2011.] http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/DEA2eTOC.html.

[9] **—. 2002.** Java BluePrints – Model-View-Controller. [Online] 2002. [Citace: 1. 5 2011.] http://java.sun.com/blueprints/patterns/MVC-detailed.html.

[10] **Tichý, Jan. 2004.** Programová podpora tvorby webových aplikací. [Online] 25. 8 2004. [Citace: 1. 5 2011.] http://www.jantichy.cz/diplomka.

[11] **Ullman, Larry. 2004.** *PHP a MySQL: Názorný průvodce tvorbou dynamických WWW stránek.* Brno : Computer Press, 2004. str. 534. ISBN 80-251-0063-4.

[12] **Vrána, Jakub. 2010.** *1001 tipů a triků pro PHP.* Brno : Computer Press, 2010. str. 456. ISBN 978-80-251-2940-1.

[13] **Welling, Luke and Thomson, Laura. 2005.** *PHP and MySQL Web Development.* Indianapolis : Sams Publishing, 2005. p. 946. ISBN 0-672-32672-8.

Příloha A – Zdrojový kód třídy FrontController

```
<?php

class FrontController implements IFrontController {

    private $actionController;
    private $actionMethod;

    public function __construct() {
        $this->setActionController();
        $this->setActionMethod();
    }

    public function delegate() {
        $actionController = null;
        $actionMethod = null;

        if ($this->actionController == null) {
            $actionController = new DefaultController();
        } else {
            $className = $this->actionController;
            $actionController = new $className();
        }

        if ($this->actionMethod == null) {
            $actionController->actionDefault();
        } else {
            $actionMethod = $this->actionMethod;
            $actionController->$actionMethod();
        }
    }

    private function setActionController() {
        $request = Request::getInstance('Request');
        $ctrl = $request->getGetValue('ctrl');

        if ($ctrl == NULL || empty($ctrl)) {
            $this->actionController = null;
        } else {
            $controllerName = $ctrl . 'Controller';
            $controllerFilePath = APP_DIR . 'controllers' . DS .
                                  $controllerName . '.php';

            if (!file_exists($controllerFilePath)) {
                $this->actionController = 'ErrorController';
            } else {
                $this->actionController = $controllerName;
            }
        }
    }

    private function setActionMethod() {
        $request = Request::getInstance('Request');
        $do = $request->getGetValue('do');

        if ($this->actionController != null &&
            $this->actionController != 'ErrorController') {
```

```
            if ($do == NULL || empty($do)) {
                $this->actionMethod = null;
            } else {
                $actionName = 'action' . $do;
                $className = $this->actionController;
                $actionController = new $className();

                if (!method_exists($actionController, $actionName)) {
                    $this->actionController = 'ErrorController';
                    $this->actionMethod = null;
                } else {
                    $this->actionMethod = $actionName;
                }
                unset($actionController);
            }
        } else {
            $this->actionMethod = null;
        }
    }

}
?>
```

Příloha B – Instalace aplikace

Na přiloženém CD se nacházejí 3 adresáře:

- Složka `application` obsahuje adresářovou strukturu aplikace se všemi zdrojovými kódy.
- Uvnitř složky `database` se nachází SQL skript (database_generate_script.sql) pro vygenerování struktury a obsahu databáze. Dále je zde soubor database_model_via_mysql_workbench.mwb, který obsahuje databázový model. Soubor je spustitelný v programu MySQL Workbench, který je volně dostupný na adrese http://dev.mysql.com/downloads/workbench/.
- Ve složce `install` je textový soubor s popisem instalace systému.

Postup instalace:

1. Obsah složky `application` zkopírovat do adresáře určeného pro běh aplikace na webovém serveru Apache.
2. Spustit SQL skript (database_generate_script.sql) v prostředí databázového systému MySQL.
3. V souborech:
   app/controllers/abstract/ABackOfficeController.php (řádek 17 a 38),
   app/controllers/abstract/AFrontOfficeController.php (řádek 17),
   app/controllers/SignOnController.php (řádek 16)
   doplnit přístupové údaje k databázovému systému MySQL.
4. V systému je automaticky vytvořen administrátorský účet s uživatelským jménem *admin* a heslem *admin*.

Aplikace je vyvíjena ve verzích MySQL: 5.1.37 a PHP: 5.2.10. Ostatní verze těchto produktů nejsou otestovány pro korektní chod aplikace.