SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Elektronická zbierka príkladov pre predmety Fyzika I a Fyzika II

BAKALÁRSKA PRÁCA

FEI-5382-17512

2011 **Andrej FARAGA**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Elektronická zbierka príkladov pre predmety Fyzika I a Fyzika II

BAKALÁRSKA PRÁCA

FEI-5382-17512

Študijný program: Aplikovaná informatika
Číslo a názov študijného odboru: 9.2.9 aplikovaná informatika
Školiace pracovisko: Katedra fyziky (FEI STU)
Vedúci záverečnej práce/školiteľ: doc. Ing. Peter Bokes, PhD.

Bratislava 2011 **Andrej FARAGA**

Slovenská technická univerzita v Bratislave
Katedra fyziky

Fakulta elektrotechniky a informatiky
2010/2011

ZADANIE BAKALÁRSKEJ PRÁCE

Evidenčné číslo:	FEI-5382-17512
ID študenta:	17512
Autor práce:	Andrej Faraga (17512)
Študijný program:	aplikovaná informatika
Študijný odbor:	9.2.9 aplikovaná informatika
Vedúci práce:	doc. Ing. Peter Bokes, PhD.
Miesto vypracovania:	Bratislava

Názov témy: **Elektronická zbierka príkladov pre predmety Fyzika I a Fyzika II**

Rozsah práce: 40

Špecifikácia zadania:

zásady nie sú zadané

Dátum zadania bakalárskej práce: **04. 09. 2009**

Termín odovzdania bakalárskej práce: **13. 05. 2011**

Andrej Faraga
Študent

prof. Ing. Julius Cirák, CSc.
Vedúci pracoviska

doc. RNDr. Gabriel Juhás, PhD.
Garant študijného programu

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

VYHLÁSENIE AUTORA

Podpísaný Andrej FARAGA čestne vyhlasujem, že som *bakalársku* prácu *Elektronická zbierka príkladov pre predmety Fyzika I a Fyzika II* vypracoval na základe poznatkov získaných počas štúdia a informácií z dostupnej literatúry uvedenej v práci.

Uvedenú prácu som vypracoval pod vedením: doc. Ing. Peter BOKES, PhD.

Bratislava, dňa 12.5.2011

.....
podpis autora

ABSTRAKT

Slovenská technická univerzita v Bratislave
FAKULTA ELEKTROTECHNIKY A INFORMATIKY
Študijný program: Aplikovaná informatika
Autor: Andrej Faraga
Názov bakalárskej práce: Elektronická zbierka príkladov z predmetov Fyzika I a Fyzika II
Vedúci bakalárskej práce: doc. Ing. Peter Bokes, PhD.
Rok odovzdania: Máj, 2011

Motiváciou práce je zefektívnenie výučbového procesu a zvyšovanie kvality a kvantity študijných materiálov. Práca sa venuje najmä návrhu a implementácii elektronickej zbierky príkladov k predmetom Fyzika I a Fyzika II. Aplikácia bude slúžiť ako vzdelávací (samovzdelávací) prostriedok študentov týchto predmetov. Práca taktiež popisuje existujúce riešenia, či už čiastkové alebo úplné a dáva obraz o stave celkovej problematiky webových aplikácií. Špeciálne časti sú venované bližšiemu popisu konkrétnych riešení, ktoré priamo alebo nepriamo súvisia s predmetom a cieľmi projektu. Údaje sú štrukturované od všeobecného prehľadu až po podrobné skúmanie špecifických skutočností. V závere sa práca venuje ďalšími možnosťami nasadenia elektronickej zbierky a spôsobmi rozšírenia za účelom dosiahnutia efektívneho a spoľahlivého výučbového nástroja.

Kľúčové slová:

elektronická zbierka, fyzika, webové aplikácie, LaTeX, HTML

ABSTRACT

Slovak University of Technology in Bratislava
FAKULTY OF ELECTRICALENGINNERING AND INFORMATION TECHNOLOGY
Študijný program: Applied informatics
Autor: Andrej Faraga
Title of bechelor project : Elektronic digest of exercises for subjects Physics I and Psysics II
Supercisor: doc. Ing. Peter Bokes, PhD.
Year of submission: May, 2011

Motivation of the project is making the learning process more effective and increasing the quality and quantity of studium materials. The work deals with blueprinting and implementation of Elektronic digest of exercises for subjects Physics I and Psysics II. Final application will serve as education (self-education) tool for students of theese subjects. The work also describes existing solutions, partial as well as complete and gives overview about curent issue's state. Special parts are dedicated to more detailed describtion of certain solutions, which are directly or indirectly related to the topic or goals of the project. The information is structured in overall overview to detailed exploration order. In the end, the work deals with alternative usage and extending of the application in order to make the learning process more effective.

Keywords:

electronic digest, physics, web applications, LaTeX, HTML

Obsah

Skratky:

ARPA	Advanced Research Projects Agency
IT	Informačné Technológie
WWW	World Wide Web
OSI Model	Open Systems Interconnection Model
HTTP	HyperText Tansfer Protocol
HTTPS	HyperText Tansfer Protocol Secure
TCP/IP	Transmission Control Protocol / Internet Protocol
HTML	Hypertext Markup Language
XML	eXtensible Markup Language
CSS	Cascading Style Sheets
PHP	Personal Home Page (server side scripting language)
SQL	Structured Query Language
W3C	World Wide Web Consortium
CRUD	Create, Read, Update, Delete
WS	Web Service

Úvod

1.1 Webové aplikácie

Webové aplikácie sú aplikácie, ktoré využívajú internet (alebo intranet) na vzájomnú komunikáciu. Ich vývoj teda úzko súvisí s rozvojom internetu. Od roku 1969, kedy vznikli prvé uzly ARPANETu, predchodcu dnešného internetu sa rozšíril ich význam do domén v ktorých predtým dominovali služby mimo oblasti IT. Papierová administratíva, „kamenné“ obchody a banky sú čoraz viac nahradzované ich elektronickými náprotivkami. Výrazy ako internet banking, e-shop, e-mail, e-learning sú toho pravým dôkazom. Práve posledný spomenutý termín, e-learning, je oblasť do ktorej možno zaradiť aj túto prácu.

Požiadavky na webovú aplikáciu sa líšia v závislosti od jej účelu. Vo všeobecnosti však platí, že jej úlohou je zefektívniť a urýchliť úkony, na ktoré aplikácia slúži. Bežné úlohy , akými sú vyhľadávanie a triedenie dát, spracovanie formulárov a rôzne matematické operácie sa stali predmetom algoritmizácie a programovania.

Existuje mnoho aspektov, podľa ktorých možno rozčleniť aplikácie do skupín. [1] Nasledujúce rozdelenie popisuje webové aplikácie podľa výpočtového modelu. Patria sem nasledujúce modely:

1. **host / terminál** – ako už naznačuje názov, aplikácia je rozdelená do dvoch celkov: host a terminál. Spracovanie a uloženie dát sa nachádza na hoste. Terminál slúži iba pre zobrazovanie údajov pre používateľa aplikácie. Používateľ zväčša zadáva príkazy na spracovanie v textovom režime a v rovnakej textovej forme vidí aj výsledky spracovania príkazu. Používateľ má k dispozícii iba časť výpočtového výkonu. Tento spôsob spracovania nie je pre užívateľa príliš komfortný.
2. **file server / klientská stanica** –. ako úložisko dát slúži file server, ktorý stanici posiela celý dátový súbor. Jeho spracovanie sa vykonáva na klientskej stanici, kde má užívateľ k dispozícii plný výpočtový výkon. Modifikovaný súbor sa posiela späť na uloženie file servera. Je teda zrejmé, že dochádza k prenosom veľkých objemov dát aj pri snahe zmeniť alebo získať iba malú časť dát.

3. **klient / server** – najrozšírenejšia forma architektúry aplikácií. Spája komfort architektúry **file server / klientská stanica** s efektívnosťou prenosu dát architektúry **host/terminál**. Spracovanie a uloženie dát a celková aplikačná logika sa nachádzajú na serveri, ktorý obsluhuje požiadavku prijatú z klientskej aplikácie. Server odpovedá správou, ktorá obsahuje iba potrebné dáta.

1.2 Architektúra klient / server

Podrobnejšie sa budem venovať práve tejto architektúre, pretože aplikácia bakalárskeho projektu využíva práve tento výpočtový model.

Celková komunikácia súčastí aplikácie pozostáva z o série požiadaviek a odpovedí na prichádzajúce požiadavky. Ide o bezstavovú komunikáciu, teda nie je možné z dlhodobého hľadiska sledovať nadviazanie ani ukončenie spojenia. Náhradným riešením je použitie premenných sedenia (session). Tieto hodnoty nie sú navzájom zdieľané medzi klientskými aplikáciami. Každá klientská aplikácia má na serveri vytvorenú vlastnú premennú sedenia (kolekciu premenných).

Tvar klientskej požiadavky, ako aj odpovede servera sa líšia v závislosti od použitého protokolu. Niektoré aplikácie vytvárajú vlastné protokoly, iné siahnu po už existujúcich spoľahlivých protokoloch. V každom prípade musí byť klientská aj serverová časť aplikácie prispôsobená na spracovanie použitého protokolu siedmej vrstvy OSI modelu – aplikačnej vrstvy. O spracovanie protokolov nižších vrstiev sa starajú zariadenia nižšej úrovne.

Výhodou použitia existujúceho protokolu je jeho dobrá podpora v sieťových architektúrach a ich kompatibilita s vybranými protokolmi nižšej úrovne OSI modelu. Najrozšírenejšia kombinácia je použitie HTTP protokolu nad TCP/IP protokolom. Mnohé aplikácie dokážu obslúžiť viac ako jeden protokol aplikačnej vrstvy (častá kombinácia HTTP a HTTPS).

1.3 Aplikácie využívajúce protokol HTTP

Protokol HTTP používa štandardný port 80. HTTPS je zabezpečená kryptovaná verzia protokolu HTTP a používa port 443. Protokoly je možné použiť rovnakým spôsobom, čo značne zjednodušuje podporu oboch protokolov v jednej aplikácii.

[2] Protokol HTTP podporuje nasledovné metódy:
GET – klient vyžaduje informáciu špecifikovaného zdroja. Požiadavka obsahujúca túto metódu by nemala mať iný charakter ako získavanie informácií
POST – odosiela dáta na spracovaniu špecifikovanému zdroju. Samotné dáta sú obsiahnuté v tele požiadavky.
HEAD – požiadavka podobná požiadavke GET. Rozdiel je v tom, že HEAD požaduje v odpovedi servera iba hlavičku bez tela.
PUT – nahrá dátovú reprezentáciu do špecifikovaného zdroja
DELETE – vymaže zdroj
TRACE – vráti naspäť prijatú požiadavku. Užitočné, keď sa chce klientský program uistiť, či nebola požiadavka modifikovaná inými sieťovými uzlami
OPTIONS – vráti zoznam serverom podporovaných metód pre špecifikovaný zdroj
PATCH – vykonáva čiastočné zmeny na špecifikovanom zdroji

Najčastejšie sú používané metódy GET a POST a sú implementované väčšinou serverov. Ostatné sú náhradne implementované na vyššej úrovni (programátorské riešenie).

Pri úspešnom nadviazaní klient/server dialógu obsahuje telo odpovede údaje, o ktoré sa klientská aplikácia zaujíma. Niektorým klientom stačia primitívne hodnoty, iné vyžadujú doplňujúce informácie o grafickom usporiadaní odpovede. Najčastejšími formátmi textových správ sú HTML a XML. Častý je aj prenos binárnych obsahov.

1.3.1 Správa vo formáte XML

[3] Správa je bez vlastnej grafickej reprezentácie. Jej úlohou je popisovať dáta. Ide o kompromis dobre čitateľného zápisu človekom aj počítačom. Typickými zástupcami klientov založených na správach XML sú čítačky RSS správ a WebService klienty.

Dokument XML sa vyznačuje použitím špeciálnych znakových konštrukcií, takzvaných tagov. Tagy rozdeľujeme na párové a nepárové. Všeobecný tvar je nasledovný:
Párový:
<meno_tagu atribut_1=”hodnota” ..... atribut_n=”hodnota”>telo</meno_tagu>
Nepárový:
<meno_tagu atribut_1=”hodnota” ..... atribut_n=”hodnota” />

Tag môže obsahovať rôzny počet atribútov. Tagy je dožné do seba navzájom vnárať takým spôsobom, že telo tagu obsauje iné tagy. Mená tagov a ich atribútov môžu nadobudnúť ľubovoľný textový tvar bez bielych znakov. Na zamedzenie konfliktov medzi rovnomennými tagmi rozlišného charakteru sa používajú menné priestory. Je to definícia všetkých možných mien tagov daného jazyka. K takejto situácii dochádza pri miešaní viacerých jazykov alebo XML správ do jedného celku. Riešenie menných konfliktov sa rieši nasledovne:
Do koreňového elementu správy sa pridá atribút *xmlns:suffix1=”url” .....*
xmlns:suffix2=”url”
Konfliktné tagy potom používajú suffix atribútu ako prefix názvu tagu:
<suffix1:konfliktny_nazov />
<suffix2:konfliktny_nazov />

1.3.2 Odpoveď vo formáte HTML

[4] Úlohou HTML je zobrazovanie dát. Popisný jazyk HTML má na rozdiel od XML sadu striktne pomenovaných tagov. Vznikla aj rozšíriteľná verzia pomenovaná XHTML (eXtensible Hypertext Markup Language). Správa písaná v HTML sa označuje ako webová stránka (webstránka). Klientská aplikácia sa nazýva webový prehliadač. Serverová časť

aplikácie sa nazýva webserver. HTML formát správy je veľmi obľúbený pre prítomnosť webového prehliadača na každom operačnom systéme.

Primárnou úlohou webového servera je hostovanie webových stránok a obsluha klientských HTTP (HTTPS) požiadaviek. Webový server môže slúžiť aj ako úložisko dát alebo ako spúšťač tzv. enterprise aplikácií.

1.3.2.1 Statické webové stránky

Stránky sa nachádzajú ako fyzické súbory na úložných zariadeniach webservra a obsahujúce celú správu vo formáte HTML. Webový server jednoducho prečíta obsah požadovaného súboru a vráti ho v tele odpovede. Odpoveď je vykreslená webovým prehliadačom do grafickej podoby. Vo všeobecnosti možno povedať, že každý prehliadač dodržuje rovnaké pravidlá pre spôsob vykreslenia. Existujú výnimky, ktoré vznikli nedokonalosťou implementácie, alebo nedodržaním všeobecných (W3C) štandardov.

Ďalšími rozšíreniami webových stránok sú okrem HTML správy kaskádové štýly (CSS) a skriptovanie na strane klienta (JavaScript). Ich kód sa môže nachádzať v samotnej webovej stránke, alebo sú to externé súbory, ktoré sa do prehliadača stiahnu v inom cykle požiadavka – odpoveď. Prepojenie s týmito súbormi je zapísané do HTML pomocou tagov. Podobným spôsobom prehliadač dopytuje obrázky použité na stránke.

1.3.2.2 Dynamické webové stránky

Enterprise aplikácie vznikli v snahe implementovať podrobnejšiu parametrizáciu požiadaviek klienta. Sú to časti programov, ktoré sa podieľajú spolu s webovým serverom na spracovaní parametrizovanej požiadavky zasielajú odpoveď šitú na mieru. Samostatné servery, ktoré sú schopné takéhoto správania sa nazývajú aplikačné servery. Je na nich možné spúšťať množstvo aplikácií (aplikácia v zmysle pokrytia celej webovej stránky v závislosti od použitej URL v požiadavke typu GET alebo POST.

Základný systém prekladu URL adresy na absolútnu cestu v súborovom systéme webového servera vyzerá nasledovne:
Klient si vyžiada zdroj na adrese
http://www.mojadomena.sk/mojastranka/priecinok/index.html.

Za predpokladu, že koreňový adresár pre hostovanie stránok v súborovom systéme je /var/www/, webový server nahradí celé doménové meno http://www.mojadomena.sk cestou ku koreňovému adresáru a prečíta súbor /var/www/ mojastranka/priecinok/index.html a vráti jeho obsah ako odpoveď. V súborovom sa nemusí nachádzať iba statický obsah ale aj súbor obsahujúci časti serverových skriptov, ktoré sa postupne spúšťajú.

Aj keď je tento spôsob prekladu URL adries najrozšírenejší, zďaleka nie je jediný. [5] Mapovanie URL adries v technológii Java Servlet Technology sa predpisuje v konfiguračnom súbore známom ako deployment descriptor pomocou mapy URL=>ServletName, kde servlet je objekt obsluhujúci prichádzajúcu požiadavku.

[6] Užitočný modul mod_rewrite na Apache servri umožňuje meniť cestu k zdrojovému súboru podľa vlastných prepisovacích pravidiel (rewrite conditions , rewrite rules).

Takýto spôsob dodatočnej konfigurácie otvára širšie možnosti pre vývojárov webových aplikácií a čiastočne zvyšuje bezpečnosť.

2 Ciele projektu

Keďže ide o aplikáciu, ktorá bude po jej vytvorení reálne nasadená pre potreby cieľovej skupiny užívateľov, treba k jej návrhu pristúpiť s ohľadom nástrahy nasadenia v produkčnom prostredí. Najpodstatnejšou úvahou je zabezpečenie postačujúcej bezpečnosti a zabrániť možnosti zneužitia aplikácie.

Ciele možno rozdeliť do týchto skupín:

1. splnenie požiadaviek školiaceho strediska
2. splnenie všeobecné požiadaviek na webovú aplikáciu
3. splnenie osobných cieľov

2.1 Požiadavky školiaceho pracoviska

Vytvorenie zadania predpokladá splnenie všetkých podmienok školiaceho pracoviska. Sú nimi najmä:

1. použitie voľne dostupných technológií – použitie balíkov softvéru, ktorými katedra disponuje, alebo má možnosť získať. Všetky súčasti by mali byť voľne dostupné (open source, freeware)
2. prispôsobenie aplikácie platforme operačného systému – použitie distribúcií softvéru, ktorú sú plne kompatibilné s touto platformou
3. schopnosť aplikácie použiť aktuálny katedrový repozitár príkladov – príklady písané v jazyku LaTeX
4. databáza príkladov – databáza s údajmi vhodnými pre zobrazenie na webe
5. možnosť siahnutia príkladov stanovených formátoch – sú to fotmáty PDF a LaTeX
6. schopnosť prispieť vlastným príspevkom vzťahujúcim sa na konkrétny príklad, možnosť vzájomnej komunikácie medzi užívateľmi, resp. medzi študentom a zadávateľom príkladu
7. implementácia interaktívny grafických apletov – pomôcok, ktoré by mali slúžiť na osvojenie si konkrétnych fyzikálnych javov a lepšie pochopenie učiva.

2.2 Všeobecné požiadavky na webovú aplikáciu

Tieto požiadavky vyplývajú najmä v dlhoročných skúseností tvorcov webových aplikácií. V priebehu rozvoja webových technológií všeobecne sa vytvorili zaužívané praktiky, ktoré sú platné dodnes. Týka sa to ako rozloženia ovládacích prvkov aplikácie, tak aj životného cyklu požiadavka-odpoveď. Umenie písať dobre formovaný dokument je kľúčovým predpokladom pre SEO (optimalizácia stránky pre vyhľadávacie stroje). Umenie vytvárať intuitívne grafické prostredie, dobré ovládanie aplikácie a umiestnenie obsahu na správnom mieste zvyšuje priazeň a spokojnosť používateľskej skupiny.

2.3 Osobné ciele

Tvorba užitočnej aplikácie ma inšpirovala. Snažil som sa pri jej tvorbe využiť poznatky, ktoré som predtým nadobudol. Zároveň som však videl možnosť vyskúšať nové princípy a technológie. Zvládnutie týchto vecí je mojim osobným predpokladom k dokončeniu aplikácie. Týmito cieľmi sú:

1. použitie architektonického vzoru MVC (Model View Controller) pre vybraný programovací jazyk
2. čiastočná personalizácia stránky - vytvorenie užívateľských skupín príkladov skupín príkladov

3 Návrh a implementácia

Pri návrhu aplikácie som sa snažil držať stanovených cieľov. Takýmto spôsobom vznikali postupne modely, na základe ktorých bolo možné uskutočniť samotnú implementáciu.

3.1 Ukladanie dát

Jednou z najzákladnejších otázok pri vytváraní dynamickej webovej aplikácie je spôsob ukladania dát. Najefektnejším riešení je použitie relačnej databázy. Inak tomu nie je ani v tomto prípade. Aplikácia musí byť schopná vykonávať štyri základné operácie nad množinou dát. Sú nimi úkony označované v skratke ako CRUD: vytváranie, čítanie, zmena, mazanie. Jazyk SQL, pomocou ktorého sa zadávajú dotazy na databázový server, ponúka široké možnosti pre ktorúkoľvek z týchto operácií. Databázový server ponúka nielen vysokú flexibilitu SQL jazyka, ale aj mnohonásobne rýchlejší prístup k dátam (v porovnaní napríklad s prácou s dátami v textových alebo binárnych súboroch).

Na základe požiadaviek katedry o voľne dostupnom softvéri ale i z vlastných dobrých skúseností som si vybral MySQL [13] server.

Prvým krokom k vytvoreniu dobrej databázy je samozrejme jej návrh. Vývojár sa musí zamyslieť nad entitami a reláciami medzi nimi a vytvoriť z nich určitý model ešte pred samotnou implementáciou. Existuje osvedčená metóda, ktorá sa skladá z troch úrovní návrhu, a tohto som sa držal pri vytváraní vhodnej databázy.

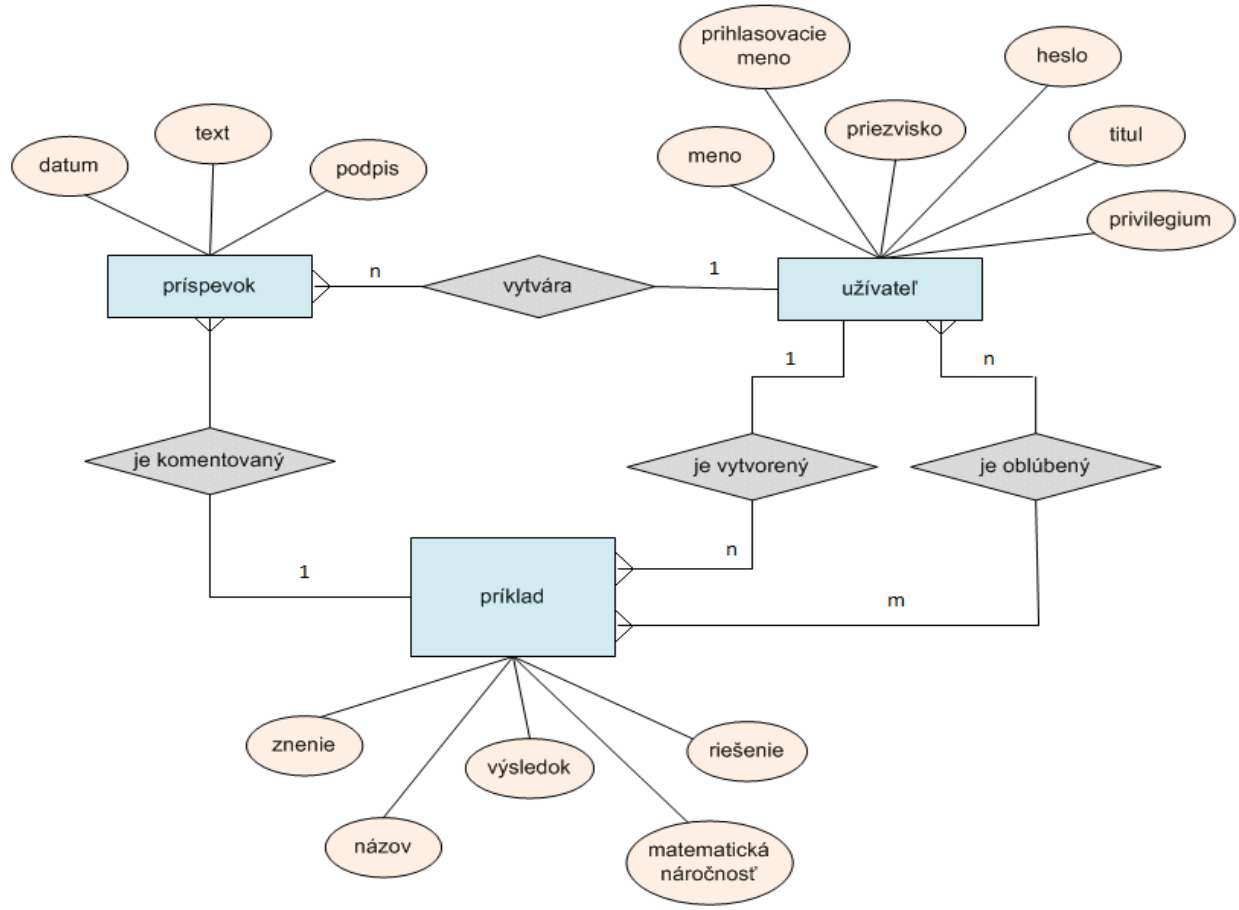
Úrovňami návrhu sú:

1. Konceptuálna úroveň
2. Logická úroveň
3. Implementačná úroveň

3.1.1 Konceptuálna úroveň návrhu relačnej databázy

[7] *„Na tejto u rovni sa snažíme popísať predmetnú oblasť pomocou všetkých entít, ktoré sa v nej vyskytujú a všetkých vzťahov medzi týmito entitami. V žiadnom prípade v tejto fáze neberieme do úvahy neskorší spôsob implementácie a do istej miery ani neskoršie obmedzenia technologického charakteru. Týmto môžeme venovať všetku energiu na pochopenie vlastného problému. Nakoniec získame aj obecne platný popis danej oblasti, ktorý môžeme použiť pre implementáciu v odlišných databázových systémoch bez nutnosti opätovnej analýzy. “*

Výsledkom tejto úrovne je entitno-relačný model graficky znázornený v ERA diagrame na obrázku 1. Celému procesu predchádzali rozsiahle debaty so školiteľom projektu.



obr. 1: ERA diagram

Entity:

priklad – nesie celú informáciu i príklade. Je najdôležitejšou entitou zbierky.

komentár – obsahuje komentáre k príkladom. Môže byť vytvorený registrovaným alebo neregistrovaným užívateľom

užívateľ – tabuľka užívateľov zbierky. Používaná je najmä pre autorizáciu a autentifikáciu užívateľov

Relácie:

príklad- užívateľ - pri tvorbe príkladov je vhodné identifikovať zadávateľa. Jeden zadávateľ je schopný vytvoriť viac príkladov. V inom zmysle spája táto väzba skutočnosť, že príklad je obľúbeným príkladom užívateľa

príklad – príspevok – možnosť prispievať viacerými príspevkami k jednému príkladu

užívateľ - príspevok – užívateľ je tvorcom príspevku

3.1.2 Logická úroveň návrhu relačnej databázy

[7] *„Pre popis dát na logickej u rovni sa v relačných databázach používa tzv. relačná schéma. Relačná schéma obsahuje tabuľky vrátane všetkých ich stĺpcov. V schéme sú vyznačené primárne kľúče v tabuľkách, ale aj cudzie kľúče ako odkaz na primárne kľúče v inej tabuľke. Najdôležitejším pracovným princípom uplatneným pri tvorbe logického návrhu je normalizácia databázy. Na vzťahoch z entitno - relačného modelu sa vykonáva viacero dekompozícií, podľa toho, akú úroveň normalizácie chceme dosiahnuť. V praxi sa najčastejšie používajú prvé tri normálne formy. Databáza v tretej normálnej forme sa považuje za dobre navrhnutú. Programátor by takto nemal mať rozsiahle problémy pre tvorbu aplikácie nad takouto databázou.“*

Normálne formy

Normalizácia je činnosť, ktorá vedie k dobre navrhnutým tabuľkám. Princípy normalizácie boli overené praxou a za dlhé roky tvorby databázových aplikácii sa vypracovali postupy tvorby tabuliek, ktoré sa nazývajú normálne formy.

Prvá normálna forma

[7] *„O relácii hovoríme, že je v prvej normálnej forme, ak sú všetky jej atribúty atomické , t.j. ďalej nedeliteľné . “*

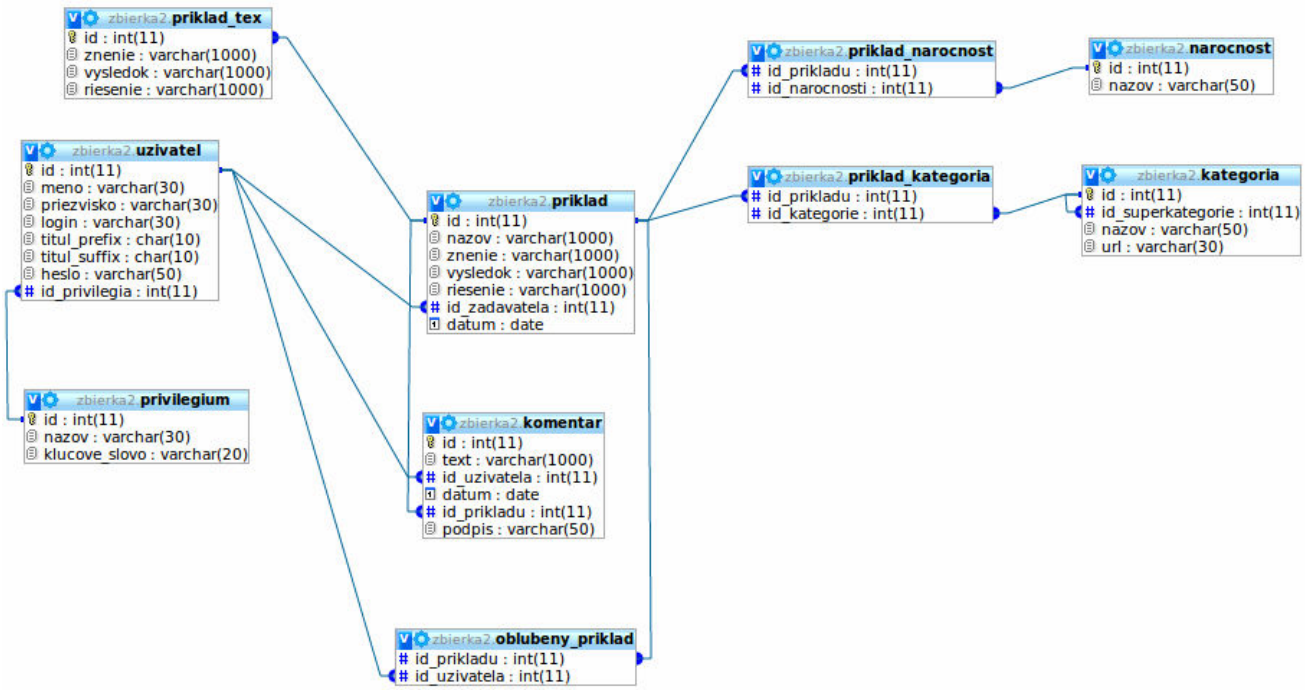
Druhá normálna forma

[7] *„Tabuľka je v druhej normálnej forme, ak je v prvej normálnej forme a naviac každý atribút, ktorý nie je primárnym kľúčom je na primárnom kľúči úplne závislý“*

Tretia normálna forma

[7] *„O relácii hovoríme, že je v tretej normálnej forme norma ak je v druhej normálnej forme a zároveň všetky jej atribúty, ktoré netvoria primárny kľúč, sú na sebe nezávislé . “*

Na samotnej databáze zbierky príkladov bolo treba vykonať niekoľko dekompozícií pre dosiahnutie tretej normálnej formy. Takýmto postupom vznikol model, ktorý popisuje schéma na obrázku č.2.



obr. 2: relačná schéma

Vykonané dekompozície:

1. Entita príklad bola rozdelená do dvoch tabuliek. Keďže samotné zadanie predpokladá ukladanie informácií matematicko-fyzikálneho charakteru v zobrazení vhodnom pre web ale aj originálneho znenia v jazyku LaTeX, boli vy tieto informácie na sebe závislé. Preto je treba vytvoriť dve verzie tabuliek (*priklady* a *priklady_tex*), z ktorých každá nesie iný tvar informácií. Sú spojené reláciou 1:1 na základe primárnych kľúčov

2. Dekompozícia atribútov príkladu do samostatných tabuliek bola vykonaná na atribútoch **fyzikálna kategória** a **matematická náročnosť**. Dôvodom bol totiž vzťah typu m:n medzi príkladom a týmito atribútmi. Vznikli celkom štyri tabuľky. Tabuľky priklad_narocnost a priklad_kategoria vznikli ako medzitabuľky ktoré majú na oboch stranách reláciu 1:n.

3.1.3 Implementačná úroveň návrhu relačnej databázy

[7] *„V tejto úrovni je potrebné zohľadniť všetky aspekty týkajúce sa použitej verzie databázového servera, platformy a je možné uplatniť isté opatrenia, ktoré sa budú týkať budúcej aplikácie, ktorá s databázou pracuje. Zmeny vykonaná v tejto fáze návrhu by nemali spätne meniť model ktorý je výsledkom doterajšej snahy.“*

Dopyty SQL, ktoré vznikli v tejto úrovni návrhu sú konštruované pre MySQL server bežiaci na platforme Unix/Linux. Keďže webové aplikácie najčastejšie používajú bezproblémovú znakovú sadu UTF-8 je táto sada aplikovaná aj na textové atribúty databázy. Stránka bude publikovať obsah v slovenskom jazyku. Je preto potrebné nastaviť systém rozlišovania hlások v dopytoch tak, aby rešpektoval slovenskú diakritiku. Pre zjednodušenie vyhľadávanie je vypnuté rozlišovanie veľkosti písma.
Kompletné nastavenie textových obsahov je: *UTF-8; Slovak; case-insensitive*.

3.2 Aplikačná časť

Rovnako ako pri riešení databázy, aj pred samotným programovaním si vývojár kladie otázku aký serverový skript použiť. Existuje viacero jazykov, pomocou ktorých je možné vytvoriť kompletnú webovú aplikáciu. Sú nimi najmä C# a technológia ASP, Java (Enterprise Edition), PHP.

3.2.1 Výber programovacieho jazyka

[8] Na programovanie aplikácie som sa rozhodol použiť PHP, kvôli nasledujúcim dôvodom:

- je licencované pod Open Source Licence – táto skutočnosť korešponduje s podmienkou školiaceho strediska o licencii použitého softvéru
- jednoduchá inštalácia – niekoľko distribúcií operačného systému Linux (vrátane distribúcie, ktorá je použitá na serveri, kde beží aplikácia) ponúka kompletný balík nástrojov pre tvorbu webových aplikácií vo svojich repozitároch. Balík zvyčajne obsahuje: webový server Apache2, PHP ako modul webového servera, MySQL server, phpMyAdmin – šikovné grafické prostredie pre administráciu MySQL databáz
- jednoduchá syntax – syntax vychádza z programovacieho jazyka C, ktorá je podobná v mnohých jazykoch. Programátorovi, ktorý má skúsenosti s iným programovacím alebo skriptovacím jazykom, to značne zjednodušuje proces učenia
- automatická typová konverzia – dátový typ premennej sa automaticky nastaví podľa literálu za priraďovacím operátorom. Ďalej je tu možnosť zmeny dátového typu premennej za behu programu, čo je v niektorých prípadoch nesmierne užitočné
- dobrá práca s poľami – PHP podporuje číselne i nečíselne indexované polia. Alternatívnu pre nečíselne indexované polia nájdeme v iných programovacích jazykoch v podobe dátovej štruktúry – hešovacia tabuľka. Nečíselne indexovaným poľom je možné simulovať a nahradiť prácu s atribútmi objektu v iných programovacích jazykoch. Ďalej PHP ponúka množstvo užitočných funkcií pre prácu s poľami.

- dobrá podpora MySQL – databázový server MySQL je v jadre podporovaný jazykom PHP. Je možné dodatočne stiahnuť knižnice aj pre iné typy databáz.
- voľné jazykové konštrukcie – v PHP je možné určitými konštrukciami docieliť vytváranie inštancií objektov a volanie funkcií bez vopred známeho identifikátora. Ako identifikátor totiž poslúži hodnota premennej typu reťazec. Obdobné riešenie nachádzame v iných jazykoch pod označením Reflection API. Jeho implementácia je však v porovnaní s PHP omnoho zložitejšia. Ukážka riešenia v PHP sa nachádza v príklade č.1.
- nie je treba kompiláciu – program sa kompiluje do strojom vykonateľnej podoby za behu programu. Táto vlastnosť prináša programátorovi viac komfortu najmä pri testovaní a viacnásobne opakovaných zmenách zdrojového kódu

Príklad č.1:

Predpokladajme, že chceme zavolať funkciu bez vopred známeho identifikátora (mena). V programe deklarujeme premennú, ktorej hodnota bude rozhodovať o volanej funkcii. Jej hodnotu inicializujeme tak, aby sa zhodovala s názvom funkcie, ktorú chceme zavolať.

```
$premenna = “skontroluj”;
function skontroluj(){
        return “skontrolované”
}
```

Funkciu **skontroluj** je potom možné volať nasledovným spôsobom:

```
$vysledok = $premenna();
```

Nevýhodou PHP je slabšia podpora OOP. Pre potreby projektu je však postačujúca a podporuje v zásade všetky princípy objektového programovania akými sú polymorfizmus, dedičnosť, zapuzdrenie, prekrývanie.

3.2.2 Životný cyklus aplikácie

Životný cyklus každej webovej aplikácie sa začína prijatím HTTP požiadavky od klienta a končí spätným odoslaním odpovede. Operácie, ktoré sa vykonajú medzi týmito hraničnými bodmi závisia od implementácie a líšia sa podľa typu a obsahu prijatej požiadavky. Dve rôzne požiadavky môžu vyvolať úplne iné časti kódu, ktoré nemajú navzájom nič spoločné. Webový server Apache2, podobne ako iné webové alebo aplikačné servery riešia výber kódu, ktorý sa má vykonať na základe URL prichádzajúcej v klientskej požiadavke. Spôsob spracovania požiadavky uvediem v príklade č.2.

Príklad č.2:

Požiadavka typu GET obsahuje nasledovnú url:
http://mojadomena.sk/priklady/index.php

1. server z URL získa podreťazec za poslednou zložkou doménového mena, teda: priklady/index.php
2. tento podreťazej je pripojený na koniec cesty ku koreňovému adresáru webových aplikácií (webroot). Za predpokladu, že webroot = „/var/www/“, dostaneme nasledovnú celkovú cestu k PHP skriptu: /var/www/priklady/index.php
3. server sa snaží zavolať kód za súboru /var/www/priklady/index.php. Ak tento súbor neexistuje, vracia odpoveď so statusom 404 (stránka nenájdená).

Keďže v takomto základnom nastavení nie je možné jednoznačne všeobecným vzorcom popísať zdrojový kód vykonávaný medzi požiadavkou a odpoveďou, prichádza otázka: „Akým spôsobom je možné volať časti zdrojového kódu v súboroch tak, aby v ňom neboli explicitne zapísané?“ PHP samo o sebe nedisponuje funkcionalitou, ktorá zabezpečuje pokročilejšie mapovanie URL adries, treba hľadať riešenie v konfigurácii samotného webového servera Apache2. Existujú dve zaužívané riešenia tohto problému:

1. auto_append_file, auto_prepend_file – použitím tejto techniky je možné automaticky volať časti kódu na začiatku (prepend) a na konci (append) kódu. Väčšinou sa používa na automatické vkladanie hlavičky a päty stránky do súboru.
2. mod_rewrite – špeciálny modul serveru Apache2, ktorý je schopný posielať požiadavku na spracovanie inému, než štandardnému PHP súboru. Nastavenie sa

vykonáva zadávaním prepisovacích pravidiel (Rewrite Rules) a podmienok (Rewrite Conditions). V týchto predpisoch sa často používajú regulárne výrazy a tým sa otvárajú nové možnosti obsluhy viacerých požiadaviek jediným kódom (jediným PHP skriptom). Veľmi často sa používa táto technika na dosiahnutie trvalých presmerovaní (response header 302).

Použitie ľubovoľnej zo spomenutých techník prináša isté úskalia pri ich aplikovaní na celú aplikáciu. Mnohokrát sa môže stať, že nastavenie nie je vhodné pre niekoľko súčastí aplikácie. Na riešenie tohto problému slúži lokálna konfigurácia častí aplikácie. Webový server Apache takúto funkciu ponúka vo forme špeciálnych konfiguračných súborov s názvom *.htaccess*. Umiestnením súboru *.htaccess* do štruktúry aplikácie zaistíme aplikovanie jeho pravidiel na obsah adresára (vrátane jeho podadresárov).

3.2.3 Segmentácia zdrojového kódu

Existuje niekoľko zaužívaných metód pre písanie prehľadného, dobre čitateľného zdrojového kódu (počínajúc konvenciami formátovania textu až po zložitejšie architektonické a návrhové vzory). Najmä pri zložitejších aplikáciách sa stáva, že programátori stratia prehľad vo vlastnom zle štrukturovanom kóde. Ak pridám úvahu, že k spolupráci pri tvorbe aplikácie pristúpi ďalší programátor bez znalosti kódu doteraz vytvorenej aplikácie, budú problémy pravdepodobne ešte väčšie. Potreba štrukturovaného kódu je v jazyku PHP o to väčšia, že celá aplikácia je uložená ako zdrojový kód (bez kompilácie). PHP dovoľuje v zdrojovom súbore miešať časti kódu technicky úplne odlišného charakteru (napríklad definície tried spolu s HTML) čo nie je najideálnejšie riešenie.

3.2.4 Architektonicný vzor MVC

Je to veľmi obľúbený spôsob segmentácie zdrojového kódu do logicky odlišných segmentov. Najčastejšie sa používa pri tvorbe webových aplikácií. Je ho taktiež možné využiť pri tvorbe desktopových aplikácií. Názov tohto vzoru vyplýva zo skratky model-view-controller. [9] Hlavnou súčasťou aplikácie sú tri druhy komponentov, každý slúži inému účelu:

1. model – doménovo špecifická sada informácií, (dát) s ktorými aplikácia pracuje
2. view – spôsob grafickej reprezentácie dát (užívateľské rozhranie)
3. controller – reaguje na akcie užívateľa, zaisťuje zmeny modelu, vykresľuje view

3.2.5 Jadro aplikácie

Jedným z osobných cieľov pri tvorbe aplikácie bolo použitie pokročilejšieho jadra aplikácie, ktoré rieši problémy týkajúce sa životného cyklu aplikácie a štruktúry kódu. Inšpiráciu pri implementácii vlastného riešenia som hľadal najmä v architektonickom vzore MVC. Keďže môže byť MVC implementované rôznymi spôsobmi, zameral som sa na jeho použitie v PHP. Vynikajúci návod sa nachádza v článkoch [10] a [11]

Výsledkom procesu implementácie je sada knižníc – zdrojových kódov v ktorých je uložená logika jadra. Tieto súbory nemusia slúžiť výhradne iba pre potreby elektronickej zbierky príkladov. Môžu byť použité ako základ inej aplikácie za predpokladu, že je konfigurácia servera Apache dostatočná a dovoľuje použiť určité techniky.
Nutnou podmienkou pre správne fungovanie aplikácie je:

- existencia a povolenie modulu *mod_rewrite*
- podpora lokálnej konfugurácie servra Apache (Allow Override) – akceptovanie súborov *.htaccess*

3.2.5.1 Životný cyklus aplikácie s použitím jadra

Nasledovné body popisujú všeobecný životný cyklus aplikácie s použitím MVC jadra pre každú klientskú požiadavku:

1. prijatie požiadavky serverom – spoločné pre všetky aplikácie
2. presmerovanie požiadavky – lokálna konfigurácia aplikácie uložená v koreňovom adresári aplikácie v súbore *.htaccess* núti server, aby namiesto zdroja požadovaného v URL adrese predal celú požiadavku na spracovanie súboru *index.php*, krorý sa nachádza taktiež v koreňovom adresári aplikácie. Realizácia tejto operácie je realizovaná modulom *mod_rewrite*. Nejedná sa o presmerovanie požiadavky. Pri presmerovaní totiž server vracia odpoveď s príznačným HTTP statusom (redirected 301, 302). Pôvodná URL z požiadavky je zachovaná ako premenná v reťazci query string, aby bola prístupná ako parameter typu *get* v zdrojovom kóde súboru *index.php*.
3. abstrahovanie údajov z požiadavky - súbor *index.php* slúži ako vstupná brána celej aplikácie. Definujú sa v ňom konštanty a vytvárajú sa objekty pre ďalšie etapy životného cyklu. Najdôležitejším objektom je *dispatcher* a jeho funkcia *dispatch*. Dochádza v nej k abstrahovaniu údajov z novovzniknutej požiadavky. Premenná, v ktorej je zachovaná požiadavka pôvodná, podlieha zmenám a rozdeľovaniu do poľa reťazcov, kde každý z reťazcov má svoj špeciálny význam
4. vytváranie inštancie príslušného *controllera* – prvým prvkom vzniknutého poľa je názov *controllera* . K jeho inštanciácii dochádza vo funkcii *Dispatcher***::***dispatch* .
5. volanie verejnej funkcie *controllera* – funkcia *handle* v tomto kroku sa stará o ďalšie spracovanie požiadavky. Jej parametrom je pole reťazcov z predchádzajúcich krokov (ale už bez názvu controllera). Prvý prvok poľa je spravidla názov privátnej funkcie *controllera*. Takto to uvádza implementácie funkcie *handle* v abstraktnej triede *AbstractController*, z ktorej dedia všetky ostatné *controllery*. Jej implementáciu je možné použitím polymorfizmu ľahko prekryť.
6. volanie spriatelenej funkcie *controllera* – trieda *contollera* môže definovať niekoľko vnútorných funkcií. Implementácia zapuzdrenia (ako vlastnosti OOP princípu) v PHP hrá v prospech bezpečnosti aplikácie. Je totiž možné volať iba funkcie deklarované ako *protected.* Takto zabezpečenú funkciu je možné vyvolať prostredníctvom funkcie

handle , pričom parametrami volanej funkcie je opäť pole z predošlých krokov, avšak už bez mena controllera a názvu spriatelenej funkcie.

7. vytváranie modelu – všetky atribúty *controllera* a všetky premenné deklarované vo funkcii môžu slúžiť ako model. Pre zamedzenie prístupu k premenným sa dá realizovať použitím inej (privátnej) funkcie *controllera.* Účelom takejto funkcie je v tomto prípade poskytovanie modelu ako návratovej hodnoty.
8. vykreslenie užívateľského rozhrania – po inicializácii premenných modelu je spravidla vykreslený komponent *view.* Je to PHP súbor, ktorý spája vypisovanie modelu so statickým obsahom (HTML, XML).

Body 7 a 8 nemusia byť v životnom cykle vôbec obsiahnuté. Príkladom môže byť spracovanie formulára, kde nie je nutné použitie komponentu *view.* Iné stránky naopak nemusia používať model v prípade, že ponúkajú iba statický obsah. Príklad spracovania konkrétnej požiadavky uvediem v nasledujúcom príklade č.2. Konkrétny príklad demonštruje vymazanie príkladu z pracovného hárku užívateľa.

Príklad č.2:

1. Server príjme požiadavku typu GET obsahujúcu URL: http://zbierka/pracovny-harok/vymaz-priklad/12
2. Transformovaná požiadavka je predaná súboru *index.php* v nasledovnom tvare: http://zbierka/index.php?url= pracovny-harok/vymaz-priklad/12
3. z *url* je vytvorené pole {PracovnyHarok , vymazPriklad , 12}
4. vytvorená inštancia objektu PracovnyHarokController , v prípade neúspechu je vytvorená inštancia ErrorController-a
5. je zavolaná funkcia *handle* (väčšinou zdedená) so skráteným poľom v parametri: handle{vymazPriklad , 12}
6. je zavolaná funkcia *vymazPriklad* so zvyškom poľa v parametri, v tomto prípade *vymazPriklad({12})*

3.2.5.2 Spracovanie údajov z formulára

Údaje z jednotlivých polí formulára sú v PHP dostupné ako pole s identifikátorom $_POST. Pole formulára s atribútom name="priezvisko" je v kóde spracovávajúceho skriptu dostupné ako $_POST['priezvisko']. Premenná post nie je inicializovaná v prípade, že ide o HTTP požiadavku typu GET. Na základe takéhoto princípu je možné rozlíšiť, o aký typ požiadavky ide.

Controller môže definovať viacero funkcií pre spracovanie prichádzajúcej požiadavky. V princípe možno poslať controlleru požiadavky typu GET i POST. K ich rozlíšenie je ponechané na implementácii spracovateľskej funkcie. Dosahuje sa to jednoduchým vetvením, preto som nepokladal za dôležité vytvoriť materskú triedu formulárového controllera, ako je to u niektorých ďalších webových aplikačných rámcov typu MVC.

Takáto slabá viazanosť zdrojového kódu na typ požiadavky má tiež svoje výhody. Predstavme si, že máme webovú aplikáciu, ktorej predmetom je trebárs kniha. Aplikácia by mala vykonávať nad knihou všetky základné operácie: vytvorenie, čítanie, zmena, vymazanie. Je zrejmé, že niektoré operácie vyžadujú spracovanie formulára, iné nie. Je výhodné aby sa o všetky operácie staral jeden controller a nie štyri nezávislé controllery. Vytváranie veľkého množstva controllerov má za následok zneprehľadnenie aplikácie a používanie dlhých názvov controllerov (napr. VymazKnihuContoller, DetailKnihyController).
Controller starajúci sa o administráciu knihy by mal obsahovať štyri dostupné funkcie:

Akcia	**Funkcia**	**URL**	**Význam**
čítanie	KnihaController::detail(20)	/kniha/detail/20/	zobrazí detail knihy s identifikátorom 20
mazanie	KnihaController::vymaz(20)	/kniha/vymaz/20/	vymaže knihu s identifikátorom 20
zmena	KnihaController::zmen(20)	/kniha/zmen/20/	vykoná zmeny na knihe s identifikátorom 20
vytvorenie	KnihaController::vytvor()	/kniha/vytvor/	vytvorí novú knihu

Existujú prípady, kedy by jedna funkcia mohla spracovávať dva druhy požiadaviek. Takáto situácia nastáva v prípade spracovania formulárov. Prvým krokom je logicky zobrazenie formulára. Prichádza teda požiadavka typu GET na to, aby bol formulár zobrazený. Nasleduje vyplnenie formulára a jeho odoslanie. Atribút *action* možno nastaviť na tú istú URL za predpokladu, že programátor implementoval rozlišovanie typu požiadavky. Odmenou bude menej zbytočných URL, ktoré sú iba technického charakteru. Táto úvaha vznikla inšpirovaním sa funkciou aplikačného rámca Spring MVC pre platformu JavaEE.

3.2.5.3 Ošetrenie chybových scenárov

Do tejto kategórie patria chyby vzniknuté odoslaním nevhodnej požiadavky na server. Štandardne je úlohou webového servera, aby pri vzniknutej chybe informoval užívateľa, o aký typ chyby ide. Protokol HTTP má v hlavičke priestor pre takzvané HTTP statusy. Pri naplnení určitého scenára vracia server odpoveď s náležitým statusom v hlavičke.

V prípade tejto aplikácie sa ošetrenie chýb značne líši. Keďže každá požiadavka je predaná na obsluhu súboru *index.php* v koreňovom adresári aplikácie, tak z hľadiska webového servera nedochádza k chybám typu 4xx alebo 5xx. Server teda v nijakom prípade nevráti odpoveď s príslušným statusom v hlavičke. Napriek tomu je potrebné takéto stavy ošetriť. Dôvodom je možnosť, že v aplikácii sa nenachádza príslušný controller alebo funkcia controllera obsluhujúca požiadavku. V oboch prípadoch dochádza k chybe so statusom 404. Tieto dva chybové scenáre sú ošetrené už v samotnom jadre aplikácie.

Na spracovanie chýb sa používa ErrorController. Parametrom jeho konštruktora je číselná hodnota reprezentujúca daný chybový typ. Od tohto typu závisí chybová správa aj samotný status v hlavičke HTTP odpovede. V prípade nenájdenia pravého controllera alebo funkcie obsluhujúcej požiadavku, vytvorí sa inštancia chybového controllera a vykoná sa jeho jediná funkcia *index()*. V tejto funkcii je možné taktiež vykresliť view chybovej stránky. V zbierke táto stránka obsahuje záhlavie spoločné pre všetky stránky, chybovú správu a spoločnú pätu. Užívateľ teda nie je nútený stlačiť tlačidlo „späť“ v ovládacom paneli prehliadača, namiesto toho môže pokračovať v browsovaní stránky pomocou hypertextových odkazov v záhlaví.

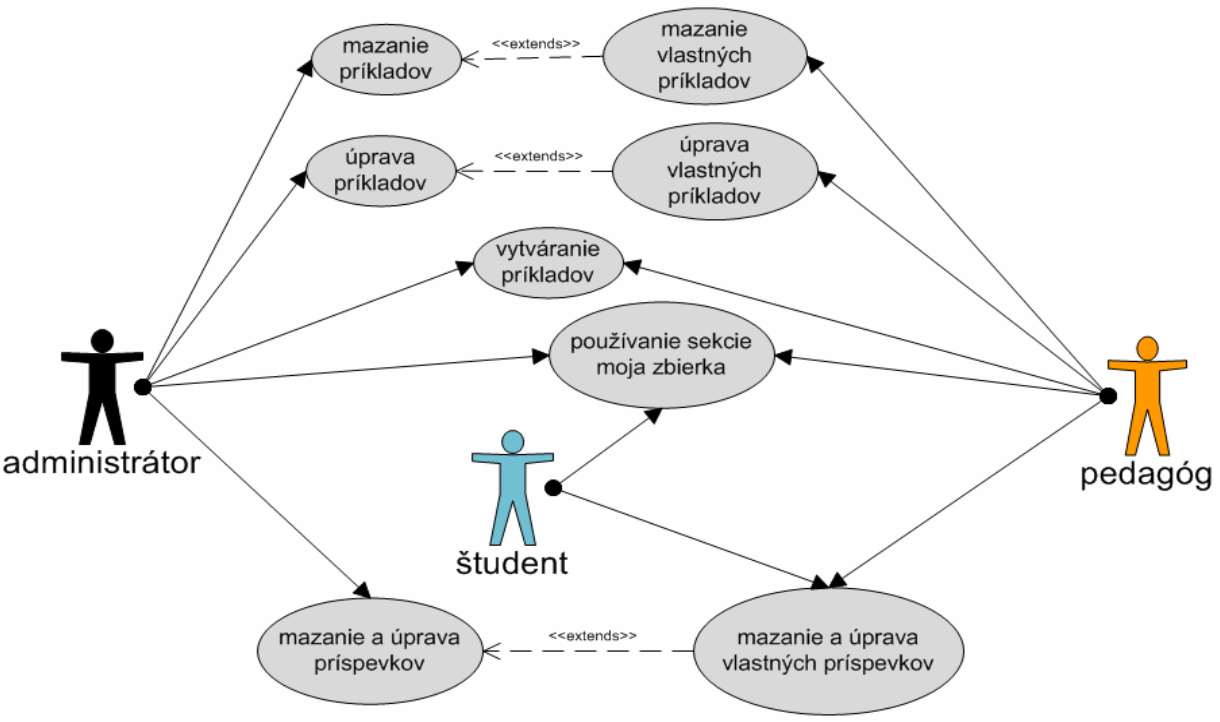
Chyba pri prehliadaní nemusí nutne nastať iba z dôvodov spomenutých vyššie, teda nemusí byť ošetrená jadrom aplikácie. V prípade, že URL adresa neobsahuje vzhľadom na existujúcu obslužnú funkciu potrebný počet parametrov, je chyba ošetrovaná z kódu tejto funkcie. Materská trieda všetkých controllerov, AbstractController, implementuje šikovnú funkciu *error(status)* , ktorý vytvorí inštanciu chybového controllera, vyvolá jeho funkciu a ukončí beh programu. Rovnakým spôsobom sa ošetrujú chyby nepovoleného prístupu (status 403). Ten nastane v prípade, ak sa neprivilegovaný užívateľ chce dostať k zabezpečenej funkcii.

3.2.6 Autentifikácia a autorizácia užívateľov

Úlohou týchto operácií je, aby mal každý užívateľ prístup iba k tým prostriedkom, na ktoré má prístupové práva. V rámci aplikácie sú vymedzené užívateľské skupiny, takzvané role. Teória prístupov je zakomponovaná v samotnom návrhu. V tejto časti sú vymedzené štyri role. Uvádzam ich v zozname v poradí od najviac privilegovanej, po rolu s najmenšími právami:

1. **Administrátor** – právo prezerať, mazať, upravovať, vytvárať všetky zdroje v rozsahu, aké aplikácia dovoľuje.
2. **Pedagóg –** jeho úlohou je vytvárať príklady a starať sa o serióznu komunikáciu s inými užívateľmi prostredníctvom písania komentárov k príkladom. Pedagóg má právo mazať a upravovať iba tie príklady, ktoré sám vytvoril.
3. **Študent –** má právo používať niektoré nástroje stránky, napríklad vytvárať vlastné kolekcie z existujúcich príkladov. Nemá právo meniť, mazať vytvárať žiadne príklady.
4. **Návštevník** – za návštevníka je pokladaná každá neprihlásená osoba. Jediný spôsob ako môže táto rola zasiahnuť do obsahu stránky, je napísanie anonymného komentára k príkladu

Podrobnejší prehľad o funkciách vyžadujúcich autorizáciu je v use-case diagrame na obrázku č.3



obr. 3: use-case diagram

Registrácia je činnosť, pri ktorej si potenciálny užívateľ vytvára svoj účet v rámci aplikácie. V aplikácii som túto časť implementoval spracovaním registračného formulára. Okrem osobných údajov a hesla, spoločných pre každý užívateľský účet, je vo formulári zahrnutý overovací kód. Overenie tohto kódu slúži na pridelenie vyšších privilégií, ktorými disponujú role pedagóg a administrátor. Samotný overovací kód je uložený v dátovej tabuľke *rola* ako atribút *overovaci_kod* a je spracovaný algoritmom MD5. Po úspešnom vytvorení účtu je do databázy vložený záznam (tabuľka *uzivatel*) s prihlasovacími a osobnými údajmi. Rola študenta nepotrebuje pre vytvorenie účtu zadať žiaden overovací kód.

Autentifikácia je činnosť, ktorá dokáže jednoznačne identifikovať užívateľa aplikácie. Vo webových aplikáciách zvyčajne riešená prihlasovaním sa do systému. Prihlasovací formulár obsahuje polia prihlasovacie meno a heslo. Jeho spracovanie spočíva v overení existencie

rovnakého mena a hesla v databázových záznamoch užívateľov. V prípade úspešného spracovania je vytvorená premenná sedenia (session), ktorá obsahuje pole informácií dôležitých pre ďalšie pôsobenie užívateľa na stránke ale najmä pre autorizáciu.

Autorizácia je činnosť, pri ktorej sa overujú práva potrebné pre prístup k zabezpečenému prostriedku aplikácie. V priebehu predošlej autentifikácie sa do premenných sedenia nahrá informácia o roli používateľa. Autorizácia je implementovaná v rôznych častiach kódu, ako neúplné vetvenie. V ňom sa overuje podmienka, že rola prihláseného užívateľa musí byť rovná alebo vyššia od požadovanej role prostriedku. V prípade, že táto podmienka nie je splnená, aplikácia odpovedá chybovou stránkou (volá funkciu error(403)).

3.2.7 Riešenie špecifických problémov aplikácie

3.2.7.1 Spracovanie formátu LaTeX

Najšpecifickejším problémom, ktorý bolo treba v rámci praktickej časti bakalárskej práce vyriešiť, je práca so zdrojmi písanými v jazyku LaTeX. Spracovanie takéhoto zápisu do formátu, ktorý je vhodný pre zobrazenie v prehliadači, nemá triviálne riešenie. Táto operácia je časovo i algoritmicky náročná. Našťastie existuje niekoľko riešení, ktoré je možné je možné v aplikácii použiť. Sú to konvertory medzi jazykmi LaTeX a HTML. Ich prehľad je možné získať na stránke http://enc.com.au/docs/latexhtml/.

Podľa výstupov je možné rozdeliť tieto konvertory na dve skupiny:

1. konvertory, ktoré používajú na zobrazenie zložitých matematických výrazov text formátovaný pomocou CSS a HTML – takýto výstup je nevhodný z hľadiska ukladania do databázy. Z hľadiska prehľadávania a filtrovania záznamov v dátovej tabuľke je dôležité eliminovať čo najväčšie množstvo technických výrazov (HTML tagov, CSS identifikátorov, atribútov, hodnôt). Ich odstránením však vznikne úplne nepoužiteľný výsledok. Ďalšou nevýhodou je nepekný výsledok, ktorý často neúplne alebo použitím viacerých znakov simuluje vzhľad pravej matematickej značky.
2. konvertory, ktoré nahrádzajú zložité matematické výrazy obrázkami - tento typ spracovania sa vyznačuje menším počtom použitých HTML tagov a CSS výrazov. Výsledný obrázok verne zobrazuje použitý matematický vzorec. Výsledok je vhodný pre zobrazenie v stránke.

Skupina konvertorov č. 2 spĺňa požiadavky pre použitie v aplikácii, preto boli moje ďalšie kroky orientované na testovanie konvertorov tohto typu. Kritériá pre výber najvhodnejšieho z nich boli nasledovné:

1. kompatibilita s distribúciou operačného systému Linux –čím viac distribúcií zabezpečí chod konvertora, tým lepšie. Balíky boli testované distribúciách Ubuntu a OpenSUSE.

2. udržiavanosť programového balíku – dôležitá z hľadiska podpory verzie XHTML 1.1 , v ktorej sú písané všetky stránky
3. rýchlosť spracovania vstupu a generovania výstupu
4. kvalita výstupu – potreba čo najmenšieho množstva HTML tagov

Najlepšie výsledky podľa týchto hodnotiacich kritérií dosiahol balík *tex4ht.* Tento balík obsahuje niekoľko spustiteľných binárnych súborov. Pre potrebu aplikácie však bol najdôležitejší príkaz *htlatex.* Jeho vstupom je cesta k LaTeX textovému dokumentu. Výstupom je niekoľko súborov rôznych formátov, medzi ktorými sa nachádza aj HTML dokument a obrázky použité v ňom. Formát obrázkov je PNG, čo je pre použitie v HTML dokumente veľmi vhodný formát. Ukážka výstupu sa nachádza v prílohe na strane č. (...)

V aplikácii je vytváranie nového príkladu implementované spracovaním formulára, do ktorého sa zadávajú texty vo formáte LaTeX. Akcie, ktoré sa vykonajú pri spracovávaní sú nasledovné:

1. spájanie častí LaTeX zdrojového dokumentu – údaje zadávané do formulára obsahujú iba predmetnú časť príkladu. Preto ich je potrebné spojiť so záhlavím a pätou LaTeX dokumentu. Obe časti sú inicializované v spracovateľskej funkcii. Vo formulári sa nachádzajú celkom tri polia, ktoré by sa mali spracovať ako LaTeX (znenie príkladu, výsledok, riešenie). Z hľadiska časovej náročnosti konverzie dokumentu sú všetky časti tiež spojené. Avšak medzi jednotlivé časti sa vloží vhodný oddeľovač, ktorý bude zachovaný aj v hrubom výstupe (dokumente HTML) príkazu *htlatex.*
2. zápis do dočasného dokumentu - cesta k súčasnému dokumentu je (...). Používanie identifinátoru sedenia v ceste k súboru sa používa preto, aby boli odlíšené výstupy v prípade, že túto akciu vykonáva naraz niekoľko používateľov. Mohla by totiž vzniknúť situácia, že používatelia si navzájom prepisujú súbory alebo čítajú cudzí výstup.
3. konverzia dokumentu – spustenie *htlatex* ako externého programu z kódu obslužnej funkcie. Všetky výstupné súbory sú vytvorené v mieste zdrojového súboru z kroku 2.
4. čítanie HTML dokumentu – údaje z dokumentu sú načítané do premennej obslužného kódu. Tá je ďalej rozdelená na základe oddeľovačov vložených v kroku 1.
5. čistenie HTML kódu – napriek tomu, že *htlatex* generuje relatívne dobrý výstup, je potrebné uskutočniť dodatočné úpravy. Výstupom je kompletný a validný dokument a teda obsahuje všetky potrebné časti v zmysle použitej DTD. V tomto kroku dochádza k odstráneniu prebytočných tagov a dlhých prázdnych miest

6. prelinkovanie obrázkov – jeden z povolených tagov v čistom HTML kóde je obrázok. Cestu k tomuto obrázku určuje atribút *src*. V tomto štádiu spracovanie je jeho hodnota nesprávne nastavená (rovnaký adresár ako HTML dokument), preto je potrebné ju zmeniť.
7. premiestňovanie obrázkov - konečné umiestnenie obrázkov je vo verejnom adresári aplikácie.
8. odstránenie dočasného adresára - tento adresár už viac nemá účel, preto je zo štruktúry vymazaný.

Z používateľského hľadiska je možné zvoliť dvojaké spracovanie formulára. V prípade, že si užívateľ zvolí uloženie príkladu, je predmetná časť príkladu vložená do databázy. Druhou možnosťou, je zobrazenie náhľadu spracovaného príkladu. V takomto prípade sa príklad do databázy neukladá, namiesto toho je jeho náhľad zobrazený v elemente nad formulárom. Takto sa môže zadávateľ príkladu uistiť o správnosti zobrazenia príkladu. Užitočnou vlastnosťou je fakt, že zadávateľ vidí HTML a zároveň LaTeX verziu príkladu.

3.2.7.2 Komunikácia používateľov

Jednou z požiadaviek školiaceho pracoviska je implementácia vhodného spôsobu komunikácie používateľov. Základným scenárom je komunikácia typu študent – pedagóg. Iný typ komunikácie tiež nie je vylúčený. Úlohou pedagóga je najmä poskytovanie odpovedí na prípadné otázky riešiteľov príkladov.

Komunikácia v zbierke je zabezpečená prostredníctvom komentárov k príkladom. Komentáre sú dostupné na detailnej stránke príkladu spolu s jednoduchý formulárom pre vkladanie nových komentárov. Každý návštevník stránky je schopný napísať komentár, ktorý je po úspešnom spracovaní vložený do databázy. Rozdiel v spracovaní príspevku prihláseným užívateľom a neprihláseným návštevníkom je v anonymite. Meno prihláseného užívateľa je automaticky zobrazované hlavičke príspevku.

Najvyššou autoritou v sekcii komentárov je zadávateľ príkladu. Preto môže mazať existujúce príspevky z pádnych dôvodov. Prihlásený používateľ môže mazať a meniť príspevky, pokiaľ je ich autorom.

3.2.7.3 Prezeranie existujúcich príkladov

Prezeranie príkladov je možné celkom v troch podstránkach:

1. Prehľad príkladov – sú v ňom obsiahnuté všetky príklady zbierky
2. Moja zbierka – táto podstránka je dostupná iba pre prihlásených používateľov. Obsahuje iba príklady, ktoré označil používateľ ako svoje obľúbené. Väzba používateľa a príkladu v tomto zmysle vyplýva z dátového modelu na obrázku č.2. Príklad je taktiež možné spomedzi obľúbených príkladov odstrániť. Táto funkcionalita vznikla na základe osobných cieľov o personalizácii zbierky.
3. Pracovný hárok - je podstránka, ktorá obsahuje príklady z konkrétneho výberu používateľa. Výhodou oproti filtrovanému zoznamu príkladov je, že pracovný hárok môže obsahovať heterogénne sady príkladov, ktoré nie je možné získať jediným spracovaním filtra. Na rozdiel od sady príkladov v sekcii Moja zbierka, nie je tento výber trvalo uložený v databáze. Namiesto toho sa používa ukladanie identifikátorov príkladov do premennej sedenia. Z toho vyplýva, že aktuálny pracovný hárok je platný do momentu, kedy expiruje platnosť sedenia. Táto doba je závislá od nastavenia webového servera.

Všetky tieto podstránky disponujú možnosťou zobrazenia filtrovanej podmnožiny príkladov v závislosti od toho, na ktorej stránke sa užívateľ nachádza. Filtrovacie kritériá sú uplatnené iba na množinu príkladov podstránky. To otvára nové možnosti vyhľadávania príkladov Mojej zbierky a Pracovného hárku.

Príklady je možné filtrovať odoslaním formulára, alebo kliknutím na položku stromového zoznamu kategórií. Stromová štruktúra kategórií je riešenie problému podkategórií. To znamená, že určitá fyzikálna kategória môže patriť do inej kategórie, alebo obsahovať skupinu vlastných podkategórií.

Špeciálnou podstránkou aplikácie je detail príkladu, ktorý obsahuje viac informácií o príklade. Navyše ponúka prezeranie diskusie o príklade a možnosť prispieť vlastným komentárom.

3.2.7.4 Sťahovanie príkladov

Ďalšou užitočnou vlastnosťou zbierky, ktorá je zároveň požiadavkou školiaceho pracoviska, je sťahovanie príkladov vo formátoch PDF a LaTeX. Oba formáty je možné stiahnuť jedotlivo alebo po skupinách príkladov.

V každej stránke, na ktorej sa zobrazuje skupina príkladov, sa nachádzajú ovládacie panely, ktoré umožňujú prácu so skupinou označených príkladov. Medzi ovládacími panelmi je možnosť stiahnutia označených príkladov.

Ďalšou možnosťou je stiahnutie jediného príkladu prostredníctvom ovládacieho panela umiestneného v blokovom elemente príkladu.

Poslednou možnosťou je stiahnutie príkladov obsiahnutých v pracovnom hárku.

Podobne ako pri spracovaní formulára pre vytvorenie nového príkladu, aj tu sa používajú rovnaké dočasné priečinky. Rozdiel je v obsahu tohto priečinku, čo do počtu a formátu generovaných súborov. Priebeh spracovania je nasledovný:

1. vytvorenie dočasného priečinku v súborovom systéme
2. zápis LaTeX kódu do súboru – v prípade jediného súboru sa vyberie z databázy predmetná časť príkladu vo formáte LaTeX. V prípade viacerých príkladov, dôjde k ich zreťazeniu a následnému zápisu do súboru.
3. konverzia do formátu PDF – dôjde k nej iba v prípade, že užívateľ zvolil formát PDF. Je zavolaný externý príkaz pdflatex, ktorého vstupom je zdrojový LaTeX súbor a výstupom je vygenerovaný súbor vo formáte PDF
4. stiahnutie súboru v požadovanom formáte
5. vymazanie dočasného priečinku zo súborového systému

4 Zhodnotenie výsledkov

Celkovým výsledkom praktickej časti bakalárskej práce je funkčná aplikácia webová podľa zadania. Funkčnosť aplikácie napĺňa podstatu takmer každého zo stanovených cieľov. Aplikácia je pripravená pre reálne nasadenie, čomu som sa snažil prispôsobiť všetky čiastkové riešenia.

Otvorená zostala požiadavka na implementáciu grafických apletov fyzikálneho charakteru. Túto oblasť som vynechal najmä z dôvodu nižšej priority. Preto som sa zameral na kvalitnejšie spracovanie riešení všetkých ostatných úloh. Tieto boli zvládnuté na uspokojivej úrovni, napriek tomu existuje niekoľko nedostatkov, pre ktoré som nenašiel riešenie. Pre všetky nedostatky boli o vytvorené náhradné riešenia, ktoré však vyžadujú použitie dodatočného kódu. Vo všeobecnosti to spôsobuje absencia alternatívnych PHP riešení oproti platformám ako je JavaEE.

Prvým nedostatkom je absencia napredovania HTTP požiadavky viacerými stránkami aplikácie. Alternatívnym riešením je presmerovanie stránky a premenné sú ukladané v premenných sedenia. Nevýhodou tohto riešenia je, že tieto premenné je treba po ukončení spracovania vymazať, čo vyžaduje použitie zbytočného kódu.

Ďalším nedostatkom je absencia objektu, ktorý vykoná časť kódu v prípade že sa naplní doba expirácie premenných sedenia. Táto časť kódu sa musí náhradne zavolať v časti spracovania požiadavky na miestach, ktoré si to nutne vyžadujú.

Žiadne z nedostatkov neovplyvňujú správanie sa aplikácie závažným spôsobom ide iba o kvalitu vyhotovenie programového kódu.

5 Záver

Práca sa venuje problematike návrhu a implementácie jednoúčelovej webovej aplikácie. Návrh aplikácie bol založený na všeobecne zaužívaných praktikách, ktoré vyplývajú z dlhoročných skúseností tvorcov webových aplikácií. Konkrétne časti návrhu boli prispôsobené účelu webovej aplikácie. V tomto štádiu som bral ohľad najmä na požiadavky školiaceho pracoviska, ktoré boli tlmočené školiteľom projektu.

Motiváciou, ale zároveň sťažením práce je okolnosť, že práca bude reálne používaná cieľovou skupinou, do ktorej patria najmä študenti fakulty a zbor pedagógov školiaceho pracoviska. Vzhľadom na tento fakt bolo potrebné zvýšenie opatrnosti najmä z hľadiska bezpečnosti, autorizácie a autentifikácie a ošetrenia chýb. Na návrh používateľského rozhrania boli taktiež kladené zvýšené nároky. Stránky sú intuitívne a dobre ovládateľné vďaka použitiu najmodernejších technológií a knižníc.

Aplikácia slúži jedinému účelu, no napriek tomu som sa snažil o vytvorenie jadra, ktoré ponúka širšie možnosti využitia v rôznych webových aplikáciách. Inšpiroval som sa princípmi architektonického vzoru Model – View – Controller, ktorý je práci opísaný. Jeho konkrétna implementácia bola prispôsobená technickým možnostiam jazyka PHP a webového servera Apache.

Použitie pokročilejšieho jadra má svoj význam z mnohých hľadísk. Predovšetkým však ide o dobrý programátorský zvyk a skvalitnenie procesu implementácie samotnej aplikácie. Mnohé z komponentov aplikácie sú znovu použiteľné, modulárne. Zdrojový kód je logicky oddelený, prehľadný. Toto má svoj význam pre prípadné rozširovanie a udržiavanie aplikácie.

6 Dokumentácia k používaniu a údržbe

6.1 Inštalácia

Pre inštaláciu je potrebné mať nainštalované tieto balíky:

1. Apache2 – webový server, odporúčaná verzia: Apache2
2. PHP – minimálne vo verzii 5.0, nainštalovaný ako modul servera Apache
3. MySQL server – nainštalovaný na lokálne alebo na dostupnom vzdialenom stroji
4. balík tex4ht – nainštalovaný so všetkými závislosťami

Priečinok webovej aplikácie je treba nakopírovať do webového koreňa aplikácie (webroot) nastaviť práva a vlastníctva tak, aby mal webový server plný prístup k celej adresárovej štruktúre.
Ďalej je potrebné vytvoriť databázu a importovať tabuľky z priloženého SQL súboru.
Všetky súbory sú priložené na CD

6.2 Konfigurácia webového servera

Pre správne fungovanie je potrebné nastaviť na webovom serveri niekoľko vecí:

1. povolenie lokálnej konfigurácie zo súborov *.htaccess*
2. povolenie modulu *mod_rewrite*

Všetky doposiaľ uvedené kroky sa líšia v závislosti od distribúcie operačného systému Unix / Linux. Preto je potrebné vyhľadať postupy, akými docieliť inštaláciu a konfiguráciu na konkrétnej distribúcii.

6.3 Konfigurácia aplikácie

Ďalšími nevyhnutnými krokmi sú nastavenia, ktoré sa uskutočnia vo vnútri súborov PHP v adresárovej štruktúre aplikácie:

1. *index.php* – nachádza sa v koreni aplikácie. Obsahuje dve konštanty, ktoré sú používané v celej aplikácii. Konštanta *APPROOT* je nastavená automaticky. Konštantu *WEBROOT* je potrebné prepísať v závislosti od relatívnej cesty adresára

webroot (z Apache konfigurácie) a súboru *index.php* . V prípade, že sa súbor index.php nachádza priamo v koreni, hodnotou konštanty je lomítko („/“).

2. dbLoader – nachádza sa v adresári *lib*. Je potrebné zmeniť niektoré časti pre pripojenie sa k databáze. Závisí to od umiestnenia databázy, privilegovaného používateľa databázy, jeho hesla a názvu schémy.
   nastavenie sa uskutoční v privátnej funkcii executeQuery, v častiach kódu *mysql_connect('hostname', 'username', 'password');* a *mysql_select_db('db_name');*

6.4 Adresárová štruktúra aplikácie

- controllers – priečinok, v ktorom sú uložené súbory so zdrojovými kódmi controllerov. Súčasné jadro aplikácie neumožňuje ukladanie týchto súborov do podpriečinkov v priečinku controllers
- global – verejne prístupný adresár. Dovoľuje ľubovoľnú vnútornú adresárovú štruktúru. Jeho účelom je uskladnenie CSS, JavaScript, obrázkových súborov
- temp – adresár určený pre dočasné uskladnenie súborov generovaných vonkajšími skriptmi (htlatex, pdflatex)
- views – priečinok pre ukladanie súborov starajúcich sa o grafickú reprezentáciu webovej stránky. Tu je možné vytvárať podpriečinky do ľubovoľnej hĺbky

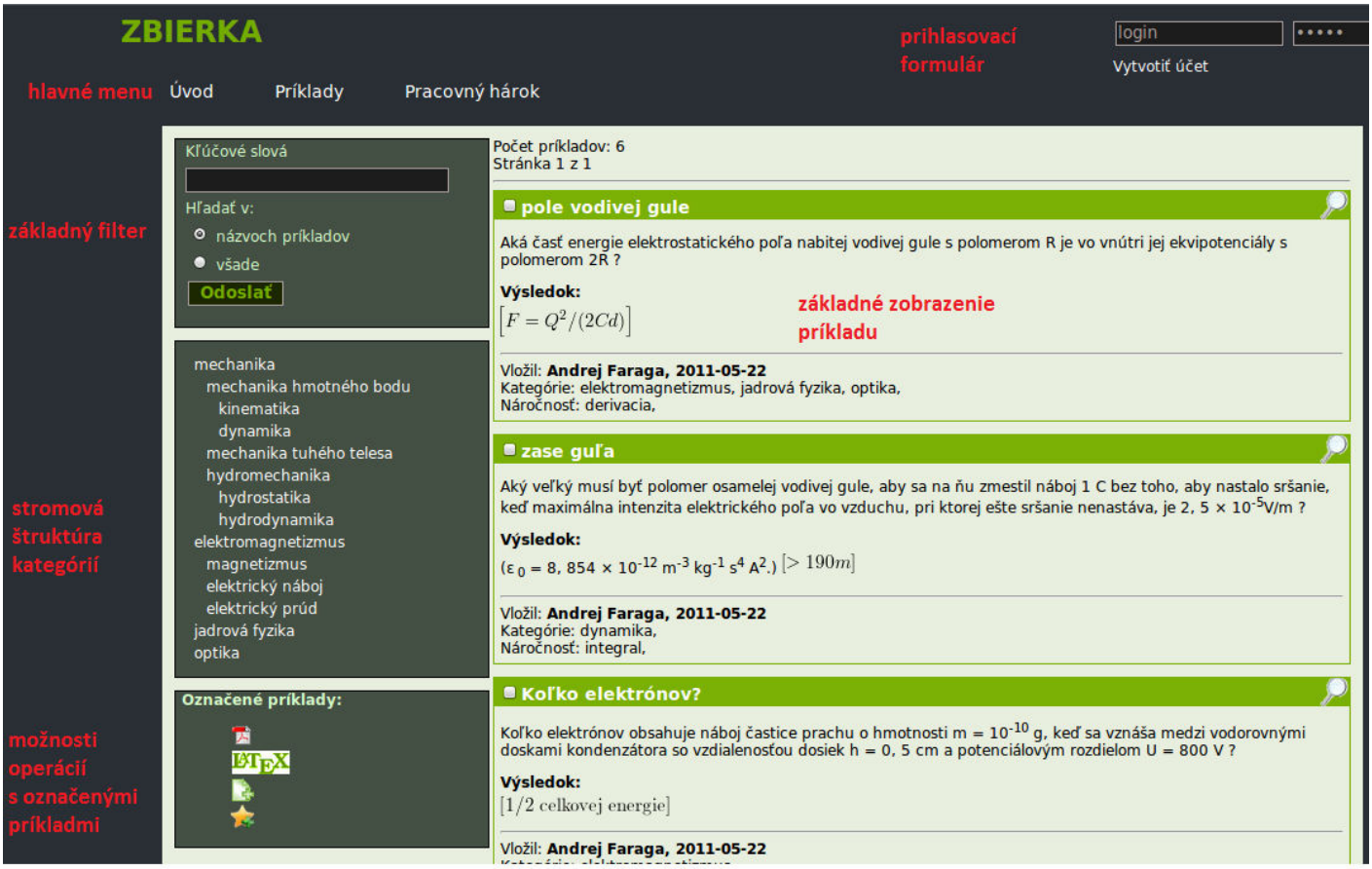
6.5 Zoznam funkčných URL adries

URL	Controller / Obsluhujúca funkcia	Význam
/, /uvod/	IndexController:: index()	vykreslí úvodnú stránku aplikácie
/priklady/	PrikladyController: :priklady()	vypíše zoznam všetkých príkladov
/priklady/{kategoria}/	PrikladyController:: priklady(kategoria)	vypíše zoznam príkladov v danej kategórii
/priklad/detail/{id}/	PrikladyController:: detail(id)	vypíše detailnú stránku príkladu konkrétneho id
/priklad/pridaj-komentar/{id}/	PrikladyController:: pridajKomentar(id)	pridá nový komentár k príkladu s konkrétnym id
/login/	LoginController:: index()	spracovanie prihlasovacieho formulára, pri úspešnom spracovaní prihlási užívateľa
/logout/	LogoutController:: index()	odhlási používateľa z aplikácie
/download/pdf/{id}/	DownloadConroller:: pdf(parameter)	stiahne príklad s id v PDF
/download/latex/{id}/	DownloadConroller:: tex(parameter)	stiahne príklad s id v LaTeX
/download/pdf/harok/	DownloadConroller:: pdf(parameter)	stiahne príklady Pracovného hárku v PDF
/download/latex/harok/	DownloadConroller:: tex(parameter)	stiahne príklady Pracovného hárku v LaTeX
/download/pdf/oznacene/	DownloadConroller:: pdf(parameter)	stiahne označené príklady v PDF
/download/latex/oznacene/	DownloadConroller:: tex(parameter)	stiahne označené príklady v LaTeX
/admin/novy-priklad/	AdminContoller:: novyPriklad()	pri požiadavke typu GET ponúkne stránku s formulárom, pri POST

		spracúva tento formulár a vkladá príklad do DB
/admin/vymaz-priklad/{id}/	AdminContoller:: vymazPriklad(id)	vymaže príklad s konkrétnym id
/admin/uprav-priklad/{id}/	AdminContoller:: upravPriklad(id)	pri požiadavke typu GET ponúkne stránku s formulárom naplnenú práve upravovaným príkladom, pri POST spracúva tento formulár a upravuje príklad v DB
/moja-zbierka/	MojaZbierkaController:: index()	vypíše príklady v sekcii Moja zbierka
/moja-zbierka/pridaj/{id}/	MojaZbierkaController:: pridaj(param)	pridá príklad s konkrétnym id do sekcie Moja zbierka
/moja-zbierka/pridaj/oznacene/	MojaZbierkaController:: pridaj(param)	pridá označené príklady do sekcie Moja zbierka
/moja-zbierka/vymaz/{id}/	MojaZbierkaController:: vymaz(param)	vymaže príklad zo sekcie Moja zbierka
/moja-zbierka/vymaz/oznacene/	MojaZbierkaController:: vymaz(param)	vymaže označené príklady zo sekcie Moja zbierka
/pracovny-harok/	PracovnyHarokContoller:: index()	vypíše príklady v sekcii Pracovný hárok
/pracovny-harok/ pridaj/{id}/	PracovnyHarokContoller:: pridaj(param)	pridá príklad s konkrétnym id do sekcie Pracovný hárok
/pracovny-harok/ pridaj/ oznacene/	PracovnyHarokContoller:: pridaj(param)	pridá označené príklady do sekcie Pracovný hárok
/pracovny-harok/ vymaz/{id}/	PracovnyHarokContoller:: vymaz(param)	vymaže príklad zo sekcie Pracovný hárok
/pracovny-harok/ vymaz/ oznacene/	PracovnyHarokContoller:: vymaz(param)	vymaže označené príklady zo sekcie Pracovný hárok

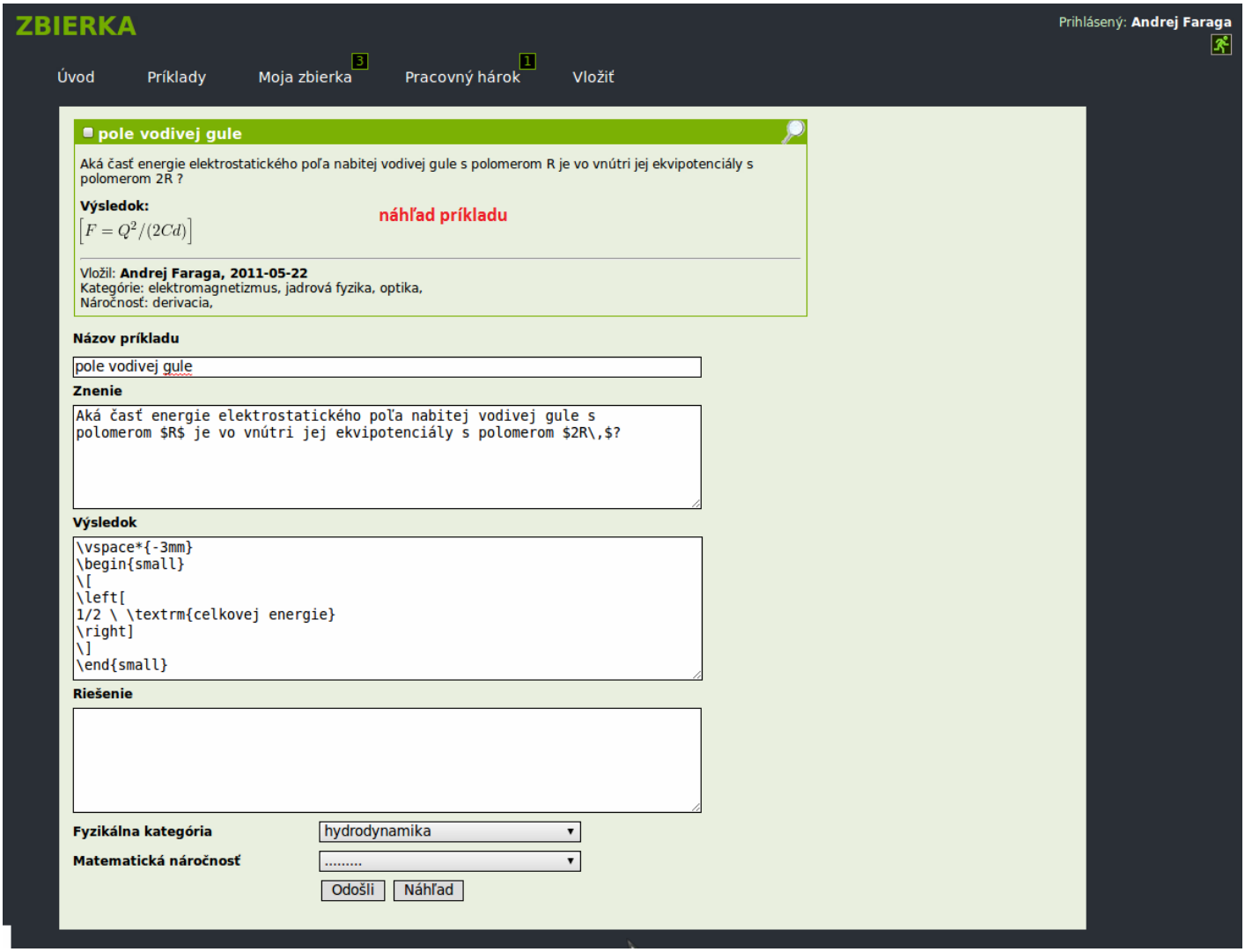
6.6 Grafické používateľské rozhranie

V nasledovných ukážkach demonštrujem funkcionalitu aplikácie a znázorním rozloženie ovládacích prvkov aplikácie.



obr. 4: rozloženie stránky 1

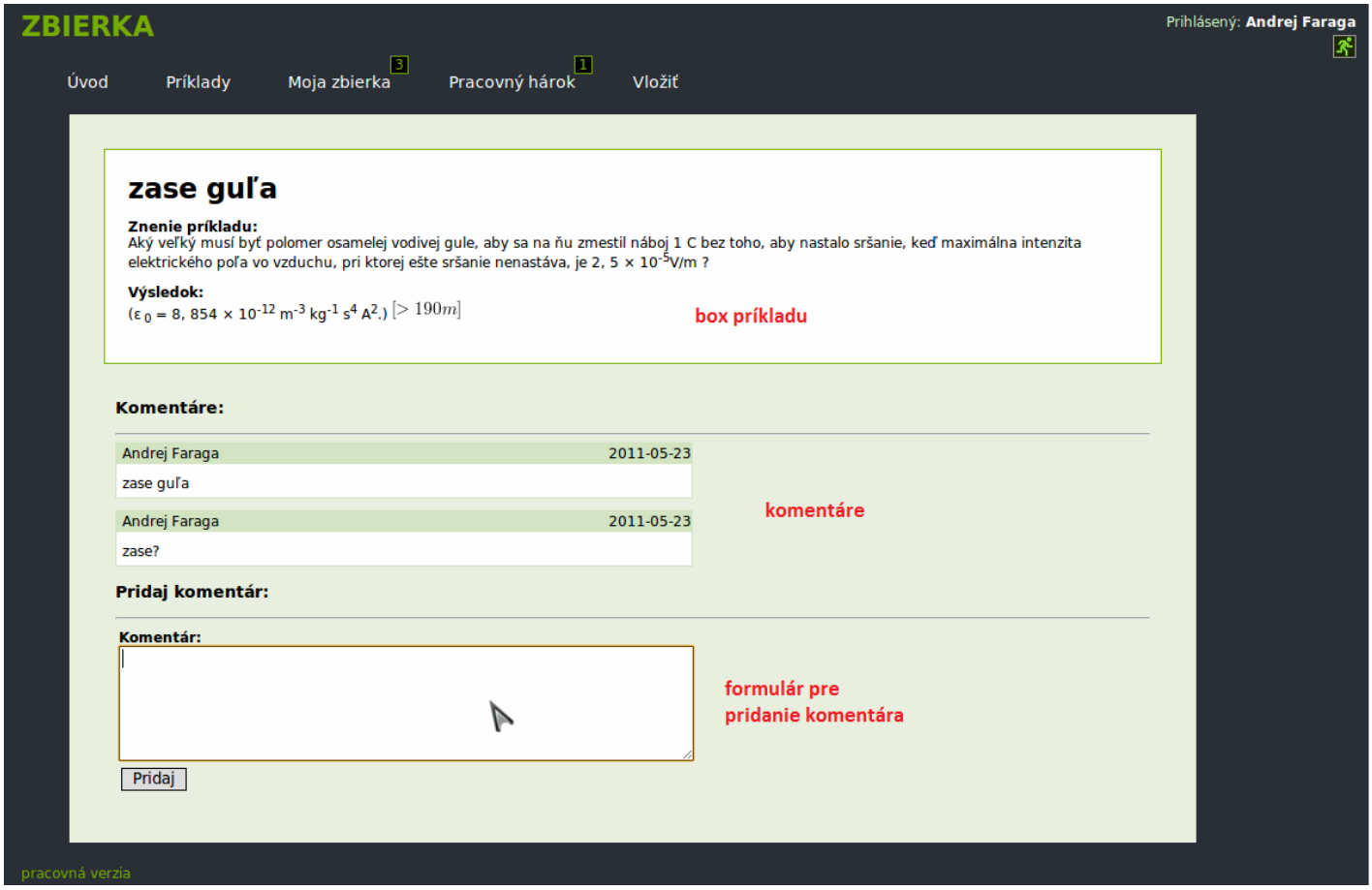
- Hlavné menu: umožňuje prechádzanie stránky pomocou hypertextových odkazov
- Základný filter: umožňuje jednoduché filtrovanie výsledkov
- Prihlasovací formulár: slúži na prihlásenie sa do systému
- Stromová štruktúra: obsahuje hypertextové odkazy, na ktorých sa zobrazia iba príklady iba príklady kategórie a jej podkategórií
- box s možnosťami operácií nad označenými príkladmi – po kliknutí vykoná danú operáciu
- základné zobrazenie: zobrazenie informácií o príklade okrem komentárov



obr. 5: rozloženie stránky 2 – formulár pre vloženie alebo úpravu príkladu

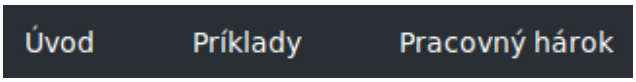
Polia formulára:

- Názov príkladu: textový názov príkladu
- Znenie: znenie príkladu vo formáte LaTeX
- Výsledok: výsledok riešenia príkladu vo formáte LaTeX
- Riešenie: popis riešenie príkladu krok za krokom vo formáte LaTeX
- Fyzikálna kategória: možnosť zvoliť fyzikálnu kategóriu príkladu
- Matematická náročnosť: výber matematických princípov uplatnených pri riešení príkladu
- tlačidlo Odošli: po úspešnom spracovaní formulára vloží príklad do databázy
- tlačidlo Náhľad: zobrazí náhľad príkladu nad poľami formulára. Neuloží do databázy.

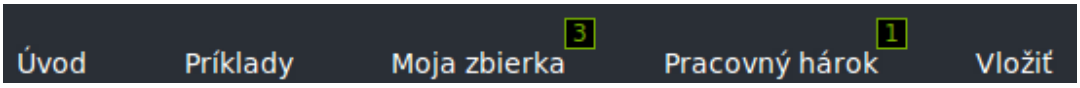


obr. 6: rozloženie stránky 2 – detailná stránka príkladu

6.7 Komponenty používateľského rozhrania



obr. 6: menu neprihláseného používateľa



obr. 7: menu vkladateľa príkladu

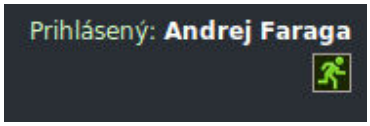
Počet a typ položiek v menu odráža úroveň privilégií prihláseného používateľa. Sú v ňom zobrazené iba také položky, na ktoré má používateľ postačujúce práva.

Položky:

- Úvod – úvodná stránka zbierky
- Príklady – zoznam všetkých príkladov
- Moja zbierka - vstup do sekcie Moja zbierka. Číslo v štvorčeku je aktuálny počet príkladov v sekcii.
- Pracovný hárok - vstup do sekcie Pracovný hárok. Číslo v štvorčeku je aktuálny počet príkladov v sekcii.
- Vložiť – vstup do stránky s formulárom pre vkladanie nového príkladu

obr. 8: prihlasovací formulár

Prihlasovací formulár ponúka možnosť prihlásenia alebo vytvorenia nového účtu.



obr. 8: odhlásenie užívateľa

obr. 9: panel nástrojov v boxe príkladu

obr. 10: panel nástrojov pre označené príklady

Tieto panely ponúkajú podobnú funkcionalitu. Rozdiel je v objekte spracovania po kliknutí na ikonku. V prípade panelu na obrázku č.9 sa spracúva jeden príklad. V paneli nástrojov z obrázku č.10 sa spracúvajú všetky označené príklady.

Typ položiek sa v závislosti od práve navštívenej stránky logicky líši. V nasledujúcej časti vypíšem všetky typy ikon, ktoré sa môžu v paneli nachádzať.

 pridať do sekcie Moja zbierka

 pridať do sekcie Pracovný hárok

 odstrániť zo sekcie Moja zbierka

 odstrániť zo sekcie Pracovný hárok

 vymazať príklad z databázy

 upraviť príklad

 stiahnuť vo formáte LaTeX

 stiahnuť vo formáte PDF

7 Použitá literatúra

1. Mrg. Miloslava Sudolská . *Architektúry spracovania údajov* [Online] 2011.
http://www.gis.sudolska.sk/architektury_spracovania_udajov.html

2. R. Fielding a spol. . *Hypertext Transfer Protocol -- HTTP/1.1* [Online] 1999
http://www.w3.org/Protocols/rfc2616/rfc2616.html

3. W3Schools . *Introduction to XML* [Online] 2011.
http://www.w3schools.com/xml/xml_whatis.asp

4. W3Schools . *HTML Introduction* [Online] 2011.
http://www.w3schools.com/html/html_intro.asp

5. Ed Ort . *Introducing the Java EE 6 Platform: Part 2* [Online] 2009.
http://java.sun.com/developer/technicalArticles/JavaEE/JavaEE6Overview_Part2.html

6. Ralf S. Engelschall . *Module mod_rewrite* [Online] 2007
http://httpd.apache.org/docs/1.3/mod/mod_rewrite.html

7. Martin Jurkovič - Miroslav Oravec . *Malé veľké databázy III* [Online] 2002
http://www.cevaro.sk/download.php?3

8. The PHP Group . *PHP Documentation* . [Online] 2011
http://www.php.net/docs.php

9. eNode . *Model-View-Controller Pattern* [Online] 2002
http://www.enode.com/x/markup/tutorial/mvc.html

10. Stanislav Riga . *Najjednoduchší MVC framework v PHP (part #1)* [Online] 2008
https://blackhole.sk/topicnajjednoduchsi-mvc-framework-v-php-part-1

11. Stanislav Riga . *Najjednoduchší MVC framework v PHP (part #2)* [Online] 2009 https://blackhole.sk/topicnajjednoduchsi-mvc-framework-v-php-part-2

12. Luke Welling - Laura Thomson . 2004 *PHP a MySQL* 1,2. Praha:SoftPress, 2004. 887. ISBN - 80-86497-60-7

13 Oracle team . *MySQL Database* . [Online] 2011 http://education.oracle.com/pls/web_prod-plq-dad/db_pages.getpage?page_id=402&p_nl=JMSQ&p_org_id=11&lang=SK