Univerzita Komenského v Bratislave

Fakulta matematiky, fyziky a informatiky

Tvorba interaktívnych webových aplikácií: prístupy, nástroje, demonštrácia

Bakalárska práca

Študijný program: Informatika
Študijný odbor: 9.2.1 Informatika
Školiace pracovisko: Katedra informatiky
Školiteľ: Dr. Tomáš Plachetka

Bratislava, 2010 **Filip Vojtko**

Rád by som sa poďakoval vedúcemu tejto práce Dr. Tomášovi Plachetkovi za pomoc, vedenie, cenné poznatky, rodine, priateľom a špeciálne mojej mame Oľge Vojtkovej za podporu a trpezlivosť. Bez týchto ľudí by moja práca nikdy nevznikla. Ďakujem.

Čestne vyhlasujem, že som bakalársku prácu vypracoval samostatne,
len s použitím uvedenej literatúry a pod odborným dohľadom školiteľa.

Bratislava, jún 2010

Abstract

Author:	Filip Vojtko
Title:	Creation of interactive web applications: approaches, tools, demonstration
University:	Comenius University, Bratislava
Faculty:	Faculty of Mathematics, Physics and Informatics
Department:	Department of Computer Science
Advisor:	Dr. Tomáš Plachetka
Thesis length:	70 pages
Bratislava, June 2010	

This bachelor thesis deals with creation of web applications from their design to testing and documentation. Goals of this thesis are to study approaches to creation of web applications, present tools supporting the development of web applications and demonstrate their use on a concrete web application.

Keywords: web application, design of architecture, security, application development, tools, team work

Abstrakt

Autor:	Filip Vojtko
Názov bakalárskej práce:	Tvorba interaktívnych webových aplikácií: prístupy, nástroje, demonštrácia
Škola:	Univerzita Komenského v Bratislave
Fakulta:	Fakulta matematiky, fyziky a informatiky
Katedra:	Katedra informatiky
Vedúci bakalárskej práce:	Dr. Tomáš Plachetka
Rozsah práce:	70 strán
Bratislava, jún 2010	

Táto bakalárska práca sa zaoberá procesom tvorby interaktívnych webových aplikácií od ich návrhu až po testovanie a dokumentáciu. Cieľmi tejto práce sú naštudovať prístupy k tvorbe webových aplikácií, nástroje na ich podporu a demonštrovať ich použitie na konkrétnej webovej aplikácii.

Kľúčové slová: webová aplikácia, návrh architektúry, bezpečnosť, vývoj aplikácií, nástroje, tímová práca

Predhovor

Internet a webové stránky sa stali každodennou súčasťou moderného života. Používame ich na svoju prezentáciu, prezentáciu našich firiem, zdieľanie zážitkov, fotografií, videí, nakupujeme. Webový vyhľadávač Google, Youtube, Wikipedia, internetové stránky denníkov a ďalšie aplikácie poznajú milióny ľudí a mnohí z nich ich aj denne používajú. Na stredných školách sa študenti učia programovať svoje prvé webové aplikácie vo forme statických personálnych stránok, vďaka jednoduchosti jazyka HTML si môže každý vytvoriť svoju vlastnú webovú stránku. Z prvých statických textových webových stránok sa tak za krátky čas stali rozsiahle dynamické webové aplikácie, ktoré v mnohých prípadoch dokážu konkurovať klasickým aplikáciám. Vytvoriť jednoduché statické stránky bolo ľahké, je však rovnako ľahké vytvárať aj dynamické webové aplikácie? Ak hovoríme, že webové aplikácie môžu byť alternatívou ku klasickým klientským aplikáciám, je možné pri ich tvorbe použiť rovnaké postupy? Existujú nejaké nástroje, metódy a postupy, ktoré zjednodušujú tvorbu klientských aplikácií. Existujú však aj podobné nástroje pre tvorbu webových aplikácií?

Bezpečnosť tradičných klientských aplikácií závisí od systému, na ktorom sú nainštalované. Ak počítač nie je pripojený k sieti, útočník sa k nemu musí fyzicky dostať, aby ho dokázal napadnúť. Webové aplikácie sú však súčasťou veľkej počítačovej siete, ako teda vieme zaručiť ich bezpečnosť? Vieme, že na prehliadanie webových stránok potrebujeme webový prehliadač. Existujú však nejaké pravidlá, ktoré zabezpečia ich správne zobrazenie a použiteľnosť aj v špeciálnych zariadeniach pre hendikepovaných ľudí? Na tieto, ale aj ďalšie otázky, sa pokúsime nájsť odpoveď.

Obsah

Zoznam obrázkov

Úvod

Prínos tejto práce

Cieľom tejto bakalárskej práce je aplikovať na tvorbu webových aplikácií vybrané poznatky z oblastí softvérového inžinierstva, projektového manažmentu, databázových systémov, bezpečnosti informačných systémov a webdizajnu. Práca má uviesť čitateľa do problematiky tvorby interaktívnych webových aplikácií a aj za pomoci názorných príkladov z tvorby reálnej webovej aplikácie čitateľa oboznamuje s riešeniami najčastejších problémov, ktoré sa môžu vyskytnúť pri tvorbe jeho vlastnej webovej aplikácie. Práca si nekladie za svoj cieľ oboznámiť čitateľa s konkrétnymi programovacími jazykmi, napriek tomu boli príklady vytvárané v jazyku XHTML a PHP, preto budeme predpokladať, že čitateľ je s nimi oboznámený.

Štruktúra práce

Práca je rozdelená do deviatich kapitol, ktoré tvoria ucelené časti. Po oboznámení sa s prvými dvoma kapitolami sa tak čitateľ môže venovať zvyšným kapitolám v poradí podľa svojho uváženia, odporúča sa však dodržať ich poradie. V každej kapitole (s výnimkou prvej) samozrejme nechýbajú mnohé príklady.

V prvej kapitole venovanej prehľadu základných pojmov uvedieme krátky prehľad technológií, o ktorých budeme ďalej v práci predpokladať, že sú čitateľovi známe. V ďalšej kapitole čitateľ nájde poznatky o tvorbe „základného kameňa“ každej aplikácie — o definícii problému a požiadaviek, pričom na podrobnom príklade sa oboznámi aj s našou webovou aplikáciou, na ktorej budeme v priebehu práce demonštrovať jednotlivé postupy. V tretej kapitole využijeme poznatky zo softvérového inžinierstva a predostrieme čitateľovi postupy, s ktorými sa aj na prvý pohľad zložitý návrh architektúry webovej aplikácie môže stať jednoduchým. Štvrtou kapitolou venovanou práci v tíme, riadeniu ľudských zdrojov a času uzavrieme časť venovanú počiatočným fázam.

V nasledujúcich troch kapitolách riešime postupne problematiky dizajnu we-

bových stránok, bezpečnosti a použitia databázového systému. Ôsma kapitola ponúka vybrané nástroje pomáhajúce pri tvorbe webových aplikácií a deviatou kapitolou prácu uzatvárame poznatkami o testovaní a dokumentácii.

Kapitola 1

Prehľad technológií

Prv než sa začneme venovať tvorbe webových aplikácií, definujeme v tejto kapitole niektoré základné pojmy z oblasti Internetu a webových stránok, ktoré považujeme za dôležité pre vývojára a na ktoré sa budeme v ďalších kapitolách odvolávať.

Internet. Celosvetová počítačová sieť. Na prenos dát vo forme paketov využíva TCP/IP (Transmission Control Protocol / Internet Protocol).

World Wide Web (WWW, web). Časť siete Internet poskytujúca informácie vo forme webových stránok pomocou Hypertext transfer protocol (HTTP). HTTP definuje niekoľko druhov žiadostí o poskytnutie webovej stránky, medzi najpoužívanejšie patria metódy GET a POST.

World Wide Web Consortium (W3C). Konzorcium vytvárajúce štandardy pre WWW.

Uniform Resource Locator (URL). Adresa miesta v sieti Internet.

Cookie. Malé množstvo dát odoslané z webového servera, prijaté a uložené prehliadačom webových stránok, ktoré sa pri každej požiadavke odosiela na server.

Session (Relácia). Permanentné spojenie medzi webovým serverom a klientským prehliadačom webových stránok. Relácie sú identifikované na základe identifikátora relácie predávaného pomocou cookie alebo ako časť URL.

Webová stránka. Dokument umiestnený na webe.

Webová aplikácia. Aplikácia typu klient-server v prostredí siete Internet.

Prehliadač webových stránok. Klientský softvér slúžiaci na zobrazenie webových aplikácií. Medzi najpoužívanejšie webové prehliadače v súčasnosti patria Microsoft Internet Explorer, Mozilla Firefox, Google Chrome, Opera a Safari.

Programovacie jazyky webových stránok na strane klienta. Skupina programovacích jazykov podporovaných prehliadačmi webových stránok. Medzi najpoužívanejšie jazyky patria HyperText Markup Language (HTML), Extensible HTML (XHTML), Cascading Style Sheets (CSS), JavaScript (známy tiež ako ECMAScript).

Webový server. Počítač alebo počítačový program poskytujúci webové stránky klientským počítačom cez HTTP protokol. Medzi najznámejšie webové servery v súčasnosti patria napríklad Apache HTTP Server, Microsoft Internet Information Services a ďalšie.

Programovacie jazyky webových stránok na strane servera. Skupina programovacích jazykov podporovaných webovými servermi. Medzi najpoužívanejšie jazyky patria PHP, Active Server Pages (ASP), Java Server Pages (JSP) a ďalšie.

Editor webových stránok. Program slúžiaci na úpravu webových stránok.

Kapitola 2

Definícia problému a požiadavky

V tejto kapitole sa budeme venovať počiatočnej fáze realizácie projektu — definícii problému a požiadavkám na aplikáciu. Zadefinujeme základné pojmy a vlastnosti požiadaviek, spomenieme niektoré nástroje a metódy na získanie a spracovanie požiadaviek a na modelovom príklade demonštrujeme ich použitie.
Na úvod definujeme skupiny osôb, ktoré sa podieľajú na procese tvorby požiadaviek.

2.1 Účastníci

Osoby, ktoré sa podieľajú na vytvorení a špecifikácii požiadaviek na výslednú aplikáciu, rozdeľujeme do troch veľkých skupín — zadávateľov, koncových užívateľov aplikácie a vývojárov. Tieto skupiny nemusia byť disjunktné.

Zadávatelia projektu definujú problém, ktorý má výsledná aplikácia riešiť. Prichádzajú s prvotnou myšlienkou, určujú základné obmedzenia projektu (čas, financie, rozsah) a definujú koncových užívateľov systému.

Koncoví užívatelia aplikácie sú ľudia, ktorí budú s výslednou aplikáciou pravidelne prichádzať do styku. Úlohami koncových užívateľov v tejto fáze projektu sú tvorba požiadaviek, vzhľad, správanie aplikácie, definícia vstupov a výstupov a prepojenia na existujúce systémy[1].

Vývojári sú ľudia, ktorých úlohou je realizácia požiadaviek zadávateľov a koncových užívateľov. Medzi vývojárov patria napríklad analytici požiadaviek, softvéroví architekti, programátori, testeri a ďalší.

[1] V prípade e-shopu sú koncovými užívateľmi nielen zákazníci, ktorí si tovar objednávajú, ale aj administrátori stránky a pracovníci obchodného oddelenia, ktorí objednávky z e-shopu vyhodnocujú.

2.2 Definícia problému

Úlohou aplikácie je riešiť problém. Základným kameňom tvorby aplikácie je definícia problému (známa aj ako podnikateľské požiadavky), ktorý aplikácia rieši. Definícia problému formuluje problém za pomoci jazyka užívateľa, z pohľadu užívateľa a neposkytuje riešenie alebo náznaky riešenia. Steve McConnell vo svojej knihe [McC04] posúva definíciu problému ďalej do abstraktnejšej roviny a tvrdí, že problém by nielen nemal byť vyjadrený pomocou počítačových termínov, ale najlepším riešením nemusí byť ani počítačový program[2].
Definícia problému je v priebehu projektu stála a je vhodné ju písomne zadokumentovať.

Príklad dobrej definície problému je „systém pre podporu organizácie cvičení z predmetu Úvod do databázových systémov“.

2.3 Požiadavky

Požiadavky na aplikáciu popisujú užívateľove nároky na funkcionalitu, správanie a vzhľad aplikácie. Požiadavky môžeme rozdeliť do dvoch skupín podľa typu — užívateľské požiadavky a funkčné požiadavky.

Užívateľské požiadavky slúžia na popis úloh (cieľov), ktoré aplikácia musí umožniť vykonávať konečným užívateľom systému.

Funkčné požiadavky sú požiadavky na správanie sa aplikácie, vstupy, výstupy a implementáciu konkrétnych funkcií a algoritmov.

Podobne ako definícia problému ani požiadavky by nemali obsahovať náznaky riešenia problému či technické detaily projektu, ale vymedzujú pre užívateľov a vývojárov ohraničenia na riešenie zadaného problému. Medzi základné vlastnosti požiadaviek patria úplnosť, správnosť, uskutočniteľnosť, nevyhnutnosť, priorita, jednoznačnosť a overiteľnosť [Wie99].

Úplnosť. Požiadavka obsahuje úplný popis funkcionality či správania sa aplikácie.
Správnosť. Požiadavka presne popisuje očakávanú funkcionalitu a nie je v rozpore s inou požiadavkou.
Uskutočniteľnosť. Požiadavka je splniteľná v stanovenom čase a za stanovených (systémových) podmienok.

[2] Ak existujú systémy, ktoré riešia čiastkové problémy, môže byť efektívnejšie využiť ľudskú silu spolu s týmito systémami namiesto vývoja nového softvérového produktu.

Nevyhnutnosť. Požiadavka je nevyhnutná pre korektné riešenie problému, prípadne je požadovaná zákazníkom.
Priorita. Požiadavka má pridelenú prioritu (reprezentovanú napr. číslom), ktorá určuje mieru dôležitosti požiadavky pre výsledný systém. Požiadavky s najvyššou prioritou sú v rámci implementácie spracované prednostne. Požiadavky s najnižšou prioritou sú v priebehu implementácie odkladané v záujme rýchleho splnenia požiadaviek s vyššou prioritou.
Jednoznačnosť. Požiadavka má práve jeden jednoznačný význam (výklad) pre každú zainteresovanú osobu.
Overiteľnosť. Pre každú požiadavku existuje testovací mechanizmus, ktorý overí, či je požiadavka splnená.

Na rozdiel od definície problému sú požiadavky na aplikáciu nemenné iba v ideálnom prípade. V reálnom svete sú nestále, nakoľko predstavy užívateľa o výslednej aplikácii sa môžu v priebehu projektu meniť. Každá zmena má za následok vytvorenie novej požiadavky na aplikáciu, ktorú je potrebné zapracovať do projektu. V závislosti od fázy, v ktorej sa projekt nachádza, sa tieto zmeny následne prejavia na výške nákladov potrebných na ich uskutočnenie[3]. Preto je veľmi dôležité v maximálnej možnej miere určiť, spresniť, odladiť a zadokumentovať všetky požiadavky na aplikáciu ešte pred zahájením samotnej stavby procesu.

2.4 Metódy zberu požiadaviek

Je viacero metód ako získať od zadávateľa a koncových užívateľov požiadavky. Uvedieme niektoré z nich:

Rozhovor. (konzultácia) Vývojár aplikácie komunikuje so zástupcami zadávateľov a jednotlivých typov koncových užívateľov. Jednotliví vhodne vybraní zástupcovia reprezentujú svoju skupinu a postupne vyslovujú svoje požiadavky a odpovedajú na otázky vývojára.
Sledovanie pri práci. Vývojár aplikácie sleduje vybraných zástupcov pri práci v snahe porozumieť jednotlivým postupom. Vývojár sa nezaujíma iba o samotný postup, ktorým užívateľ vykonáva danú činnosť, ale snaží sa odhaliť príčinu, pre ktorú činnosť vykonáva, čím získava ďalšie požiadavky na aplikáciu.
Pohľad z druhej strany. Jeden z najjednoduchších spôsobov, ako môže vývojár získať požiadavky, je vykonávať po určitú dobu prácu namiesto užívateľa. Vývojár získa pohľad na problematiku zo strany užívateľa a pochopí princípy a príčiny činností, ktoré užívateľ pri svojej práci vykonáva.
Prieskum trhu. S pravdepodobnosťou blízkou istote sa dá predpokladať, že už

[3]V praxi sa ukázalo, že úpravou požiadaviek po nasadení aplikácie môže dôjsť k 25- až 100-násobnému nárastu nákladov potrebných na realizáciu zmien oproti nákladom v raných fázach projektu [McC04].

existuje aplikácia, ktorá rieši rovnaký alebo podobný problém. Ďalšou z metód na získanie požiadaviek je teda odskúšanie existujúcich riešení s cieľom nájsť požiadavky, ktoré sú relevantné aj pre aktuálny projekt.
Recyklácia. Existujúce staršie projekty môžu obsahovať požiadavky, ktoré sú relevantné aj pre aktuálnu aplikáciu.

2.5 Príklad

2.5.1 Definícia problému

- Vytvorenie systému pre podporu organizácie cvičení z Úvodu do databázových systémov.

2.5.2 Užívateľské požiadavky

- Aplikácia podporuje 4 typy užívateľov: administrátor, učiteľ, študent a anonymný (neprihlásený) užívateľ.
- Užívatelia vidia aktuálne informácie.
- Neprihlásený užívateľ sa vie prihlásiť.
- Prihlásení užívatelia majú vytvorený profil, ktorého vybrané časti môžu upravovať.
- Prihlásení užívatelia majú možnosť zobrazenia zoznamu cvičení.
- Užívateľ typu študent má možnosť sa na cvičenie prihlásiť a odhlásiť.
- Užívateľ typu učiteľ spravuje pridelené cvičenia.
- Správa cvičenia je zložená zo zoznamu prihlásených študentov a jeho úpravy, správy miestnosti a času konania cvičenia.
- Užívateľ typu administrátor spravuje všetky cvičenia a všetkých užívateľov.
- Užívateľ typu učiteľ alebo typu administrátor má možnosť kontaktovať študentov.
- Užívateľ typu učiteľ alebo typu administrátor má možnosť publikovať oznamy (informácie) na web stránke.
- Užívateľ typu študent má možnosť kontaktovať svojho učiteľa alebo administrátora.

2.5.3 Funkčné požiadavky

Aplikácia

- Výstupom aplikácie je webová stránka vo formáte XHTML 1.0 Transitional sformátovaním v CSS 2.1.
- Výstup aplikácie spĺňa WCAG 2.0 AA (a vyššie).

Profil užívateľa

- Profil užívateľa obsahuje meno a priezvisko, e-mailovú adresu a študijnú skupinu (rok štúdia + skratka študijného programu).
- E-mailová adresa užívateľa je neverejná a slúži výhradne na zasielanie notifikačných správ.
- E-mailová adresa užívateľa typu študent musí byť z domény st.fmph.uniba.sk.
- Užívateľ má možnosť zmeniť svoju e-mailovú adresu, ostatné položky profilu môže upravovať iba administrátor.

Správa užívateľov

- Užívateľ typu administrátor spravuje kontá užívateľov typov študent a učiteľ za pomoci nástroja na správu užívateľov.
- Nástroj na správu užívateľov poskytuje prostriedky na vytváranie nových, úpravy a odstránenie existujúcich kont.
- Kontá užívateľov typu administrátor nie je možné upravovať ani odstrániť.
- Pridať nových užívateľov (typy študent, učiteľ a administrátor) je možné jednotlivo vyplnením formuláru alebo importovaním zo súboru vo formáte CSV.
- Odstránenie užívateľa znamená odhlásenie užívateľa z navštevovaných skupín a vymazanie užívateľského konta zo systému.

Správa skupín

- Užívateľ typu administrátor spravuje zoznam skupín cvičení.
- Skupiny je možné cez webové rozhranie pridať, odstrániť a upraviť.
- Skupina má názov, názov predmetu, čas výučby, pridelenú miestnosť a pridelených učiteľov.

- Skupinu je možné uzamknúť. Do uzamknutej skupiny sa študent nemôže prihlásiť.
- Skupinu nie je možné odstrániť, ak obsahuje aspoň 1 študenta alebo učiteľa.
- Užívateľ typov administrátor alebo skupine pridelený učiteľ spravujú zoznam prihlásených študentov v skupine.
- Na správu prihlásených študentov slúži webové rozhranie, ktoré umožňuje prihlásených študentov odhlásiť alebo ručne prihlásiť študenta do skupiny (limit študentov sa ignoruje).

Správa miestností

- Užívateľ typu administrátor spravuje zoznam miestností, v ktorých prebieha vyučovanie.
- Miestnosť je možné cez webové rozhranie pridať, odstrániť a upraviť.
- Miestnosť má názov a maximálny počet študentov.
- Miestnosť je možné uzamknúť. Uzamknutú miestnosť nie je možné použiť pre aktuálnu výučbu.
- Miestnosť nie je možné uzamknúť alebo odstrániť, ak je používaná na výučbu.

Prihlásenie sa do systému

- Užívatelia sa do systému prihlasujú pomocou prihlasovacieho mena a hesla.

Zmena prihlasovacieho hesla

- Užívateľ má možnosť zmeny svojho prihlasovacieho hesla.
- Administrátor má možnosť zmeny prihlasovacieho hesla užívateľov typu učiteľ a študent.
- Prihlasovacie heslo nie je možné spätne získať.

Prihlásenie/Odhlásenie sa do/zo skupiny

- Užívateľ typu študent sa môže prihlásiť do neuzamknutej skupiny.
- Užívateľ môže byť súčasne prihlásený maximálne do 1 skupiny.
- V prípade plnej obsadenosti skupiny má užívateľ typu študent možnosť zaradiť sa medzi čakateľov na prijatie až do 3 skupín podľa svojej priority.

- Užívateľ typu študent sa môže odhlásiť z neuzamknutej skupiny.

Správa fronty čakateľov na prijatie do skupiny

- Preraďovanie študentov medzi skupinami prebieha v pravidelných intervaloch raz za 6 hodín.
- Ak je študent prihlásený do skupiny a zároveň je zaradený ako čakateľ do skupiny (skupín) s vyššou prioritou, je v prípade uvoľnenia miesta do skupiny s vyššou prioritou preradený.
- V prípade preradenia študenta do skupiny s nižšou prioritou študent ostáva čakateľom na preradenie do všetkých skupín, ktoré majú vyššiu prioritu.
- Pri preraďovaní študentov rozhoduje čas prijatia žiadosti o preradenie.
- Študent nemôže byť preradený do skupiny, ak je skupina uzamknutá alebo bol dosiahnutý maximálny počet študentov.
- Ak študent žiada o preradenie do skupiny, v ktorej iný študent žiada o preradenie do skupiny prvého študenta, je výmena študentov uskutočnená aj v prípade, ak je jedna (alebo obe) zo skupín obsadená.
- Študent je upovedomený o zmene navštevovanej skupiny v určený čas pred začiatkom výučby v skupinách, ktorý si môže študent zvoliť. V prípade prekročenia určeného času je preradenie študenta uskutočnené po skončení výučby pre daný týždeň v oboch skupinách.

E-mailová notifikácia

- Systém by mal posielať notifikačné e-mailové správy pri vybraných udalostiach.
- Udalosti vyžadujúce notifikáciu sú: vytvorenie užívateľa, prihlásenie a odhlásenie sa z cvičenia.

Bezpečnostná politika

- Prihlasovacie heslo je zložené z alfanumerických znakov.
- Prvotné heslo vytvára administrátor.

2.5.4 Systémové požiadavky

- Aplikácia sa musí správne zobrazovať v internetových prehliadačoch Microsoft Internet Explorer 6 a vyššie, Opera 5 a vyššie, Mozilla Firefox 2.5 a vyššie.
- Aplikácia musí „bežať“ na webovom serveri Apache 2.2.15 a vyššie.
- Aplikácia musí podporovať databázové servery MySQL a PostgreSQL 8.3 a vyššie.

Kapitola 3

Návrh architektúry

„Návrh je aktivitou, ktorá spája požiadavky s kódom a s následným ladením. Dobrý celkový návrh poskytuje štruktúru, ktorá môže bezpečne obsahovať viac nízkoúrovňových návrhov. Dobrý návrh je pre malé projekty užitočný, zatiaľ čo pre veľké projekty je prakticky nenahraditeľný.“ [McC04]

V tejto kapitole v krátkosti uvedieme a popíšeme vlastnosti návrhu, uvedieme niektoré nástroje, ktoré sa používajú pri tvorbe návrhu architektúry, a názorne ukážeme ich použitie na vybraných častiach návrhu aplikácie, ktorej požiadavky sme v minulej kapitole uviedli ako príklad.

3.1 Vlastnosti návrhu

„Usudzujem, že existujú dve možnosti ako vytvoriť návrh: buď je tak jednoduchý, že je zrejme bezchybný, alebo je tak zložitý, že chyby v ňom nie sú zrejmé.“ [Hoa81]

Tvorba návrhu nie je deterministický proces — neexistuje univerzálna sada pravidiel, ktorá by návrh aplikácie zaručene viedla k optimálnemu riešeniu. Napriek tomu poznáme vlastnosti, ktoré by mal dobrý návrh architektúry poskytovať, a nástroje (metódy), ktoré tvorbu návrhu uľahčujú:

Flexibilita. Flexibilný návrh architektúry je návrh, ktorý bez ďalších významných nákladov umožňuje aplikáciu prispôsobiť aktuálnej situácii. Návrh sa tak stáva odolnejším voči dodatočným požiadavkám zákazníka a vývojárom napríklad umožňuje vo veľmi krátkom čase zmeniť použitý databázový systém, užívateľské rozhranie či integrovať iné produkty do aplikácie.
Jednoduchosť a zrozumiteľnosť. Jedným z primárnych cieľov pri vytváraní návrhu je snaha o vyhýbanie sa zložitým (komplikovaným) návrhom. Zložité návrhy nielen vyžadujú dôkladnejšiu a časovo náročnejšiu prípravu, ale najmä zhoršujú zrozumiteľnosť návrhu pre ďalšie osoby, ktoré môžu prísť do styku s

návrhom v budúcnosti[1]. Snahou vývojára je teda vytvoriť samopopisný návrh.
Opätovná použiteľnosť. Vývoj jednotlivých súčastí je časovo (i finančne) náročný. Vývojár by si mal byť tejto skutočnosti pri návrhu súčastí aplikácie vedomý a súčasti navrhovať tak, aby boli opätovne použiteľné aj v iných subsystémoch, resp. v budúcich aplikáciách.
Ortogonalita. Ak zmeny v niektorej časti aplikácie nemajú vplyv na inú časť aplikácie, hovoríme, že tieto dve časti sú ortogonálne. Hlavnými výhodami ortogonálnych systémov sú zrozumiteľnosť a izolácia chýb. Jednotlivé časti (komponenty) aplikácie navzájom komunikujú prostredníctvom pevne stanovených rozhraní a navonok predstavujú tzv. „čierne krabice“[2]. Čierne krabice je možné vytvoriť zapúzdrením objektov[3].
Prenositeľnosť. Aplikácia je v maximálnej možnej miere platformovo nezávislá. Prípadná zmena technológií tak v budúcnosti v dôsledku technologického vývoja, preferencií užívateľa alebo iných faktorov, nevyžaduje významný zásah do aplikácie. Jednou z veľmi častých zmien technológií je výmena databázového systému v dôsledku zmeny webhostingu užívateľom. Ak vývojár nemyslí na prenositeľnosť, použije jazyk PHP s metódami mysql_connect, mysql_query, atď. pre databázový systém MySQL a pre MySQL špecifickými SQL dotazmi, stratí značné množstvo času prepisovaním kódu aplikácie pri každej zmene databázového systému zo strany užívateľa, pričom danej situácii sa je možné vyhnúť správnym návrhom aplikácie[4].
Rozšíriteľnosť. Dobrý návrh aplikácie uvažuje s možným rozšírením funkcionality aplikácie v budúcnosti a vývojárom umožňuje rozširovať funkcionalitu vybraných subsystémov bez významných zásahov do iných častí aplikácie aj po jej nasadení do ostrej prevádzky.
Štíhlosť. Aplikácia je navrhovaná tak, aby neobsahovala žiadne prebytočné časti oproti požiadavkám. Každá prebytočná časť so sebou prináša nevyhnutnosť tvorby nového kódu, testovania a zvyšuje náklady na údržbu či prípadnú zmenu aplikácie v budúcnosti. „Dokonalosť nie je dosiahnutá vtedy, keď už nie je čo pridať, ale vtedy, keď už nie je možné nič odstrániť.“ |Antoine de Saint-Exupéry|
Štandardné techniky. Vyhýbanie sa exotickým postupom, nástrojom a využívanie štandardných techník pri tvorbe aplikácie zjednodušujú porozumenie pre vývojárov, ktorí budú do projektu zapojení v neskorších fázach.

[1] Najhorším príznakom zložitej a nezrozumiteľnej architektúry je zaskočený autor návrhu, ktorý má už aj po krátkom čase od vytvorenia návrhu problém s uspokojivými odpoveďami na otázky ohľadom niektorej jeho časti.

[2] „Čierna krabica“ je metóda na skrytie vnútornej štruktúry a implementácie pred používateľom „krabice“. Požívateľ (program alebo človek) pozná funkciu, vstupy a výstupy krabice, ako to krabica „robí“ je pred ním utajené.

[3] Napríklad pomocou návrhového vzoru Most, ktorý spomenieme neskôr.

[4] Viď nižšie Návrhové vzory - Most.

3.2 Odstupňovaný návrh

„Zdá sa, že hierarchické (odstupňované) systémy majú tú vlastnosť, že niečo považované za nedeliteľnú časť na jednej úrovni, je považované na nižšej úrovni s väčšou mierou detailov za objekt zložený. Výsledkom je, že prechodom uvažovania z vyššej úrovne na nižšiu rádovo klesajú rozsah a čas potrebné na spracovanie každej úrovne[5].“ [Dij72]

Odstupňovaný návrh je spôsob návrhu, pri ktorom je aplikácia rozdelená do niekoľkých úrovní abstrakcie (vrstiev), pričom každá úroveň je zložená z komponentov najbližšej nižšej úrovne, s ktorými komunikuje pomocou pripravených rozhraní. Cieľom je teda vytvoriť jednoduchý a zrozumiteľný návrh, ktorý sa v maximálnej možnej miere snaží o dodržanie ortogonality medzi jeho jednotlivými časťami. Ukazuje sa, že takýto spôsob návrhu zároveň umožňuje pomerne ľahko dosiahnuť aj ďalšie vlastnosti, preto teraz uvedieme jednotlivé vrstvy ako ich popisuje Steve McConnell [McC04] a demonštrujeme ich použitie na našej aplikácii.

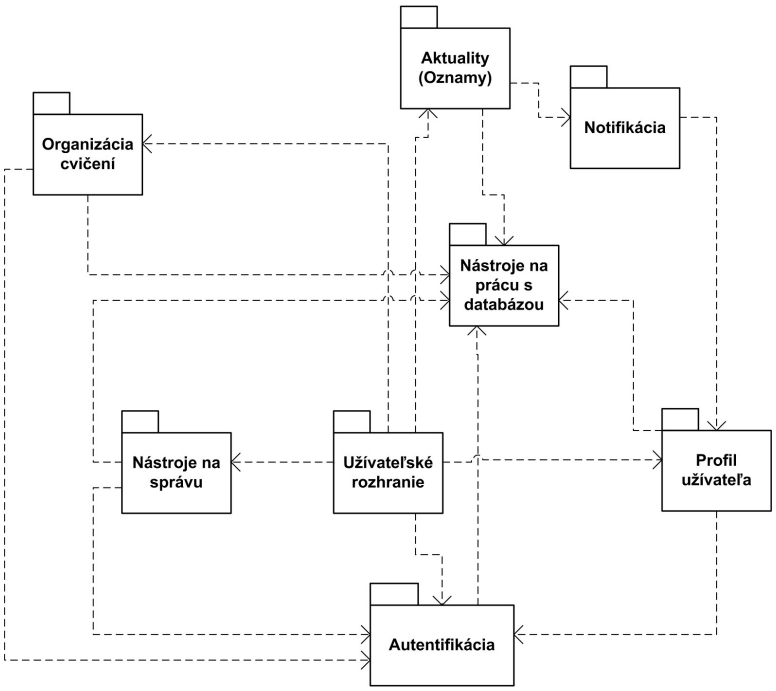
3.2.1 1. vrstva: Aplikácia

1. vrstva návrhu je tvorená výslednou aplikáciou. Aplikácia sa delí na jeden alebo viac balíčkov (modulov).

3.2.2 2. vrstva: Balíčky (subsystémy)

2. vrstva návrhu obsahuje zoznam všetkých balíčkov, popis balíčkov, vzájomnú komunikáciu medzi balíčkami a ich použitie v aplikácii.
Balíček je úplne samostatný, ak pre svoju funkčnosť nepožaduje komunikačný kanál s iným balíčkom (alebo jeho triedami). Pri návrhu balíčkov je kladený dôraz na ich maximálnu samostatnosť. V reálnom návrhu existujú balíčky, ktoré nie je možné navrhnúť ako úplne samostatné — vyžadujú komunikáciu s inými balíčkami (balíček užívateľského rozhrania vyžaduje informácie z balíčka správy databázového systému). Ak existujú balíčky, ktoré nie sú úplne samostatné, navrhujú sa v tejto vrstve aj komunikačné zásady. Komunikačné zásady sú pravidlá, ktoré obmedzujú použitie komunikačných kanálov. Medzi komunikačné zásady patrí napríklad zákaz volania interných tried a metód iných balíčkov (balíček volá mimo vlastných tried a metód iba metódy komunikačného rozhrania iného balíčka) alebo minimalizácia celkového počtu komunikačných kanálov (užívateľské rozhranie nepotrebuje nutne vlastný komunikačný kanál pre priamy prístup k databáze, potrebné informácie z databázy sprostredkuje napríklad riadiaci balíček s aplikačnou logikou). Balíčky (subsystémy) sú tvorené triedami.

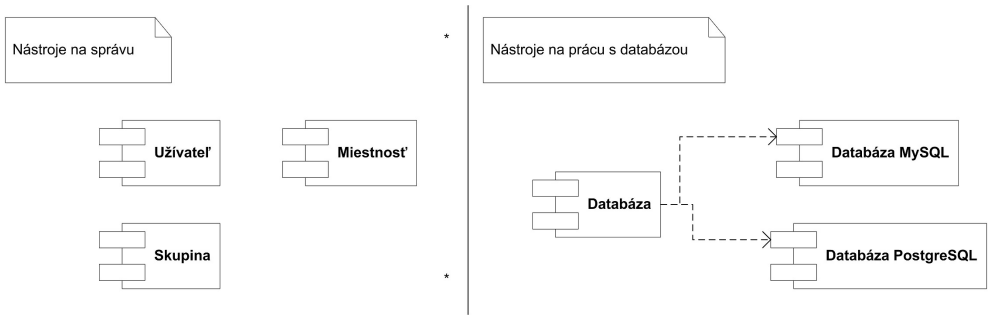
[5] Múry sa skladajú z tehál, tehly z častíc hliny, častice z molekúl atď.



Obr. 3.1: Balíčky v systéme pre podporu organizácie cvičení z Úvodu do databázových systémov

3.2.3 3. vrstva: Triedy

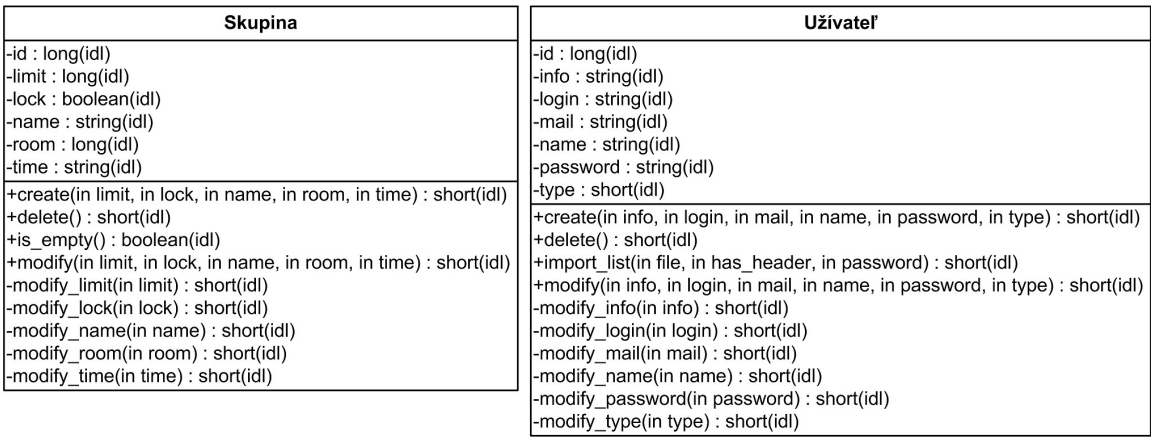
3. vrstva návrhu obsahuje zoznam všetkých tried v rámci jednotlivých subsystémov. Na tejto úrovni je každej triede navrhnuté rozhranie, ktoré umožňuje komunikáciu triedy s okolitým svetom. Rozhranie triedy je navrhnuté formou verejných metód, premenných a vlastností tak, aby bolo nezávislé na implementačných detailoch triedy. Pri návrhu rozhrania môžeme využiť návrhové vzory (napríklad návrhový vzor fasáda alebo most), ktoré spomenieme neskôr. Triedy sa ďalej delia na dáta a metódy.



Obr. 3.2: Triedy v balíčkoch Nástroje na správu a Nástroje na prácu s databázou

3.2.4 4. vrstva: Dáta a metódy

4. vrstva návrhu obsahuje detaily o delení každej triedy. Každá trieda má názov, dátovú časť (premenné) a metódy (funkcie, procedúry). Premenné a metódy môžu byť statické alebo dynamické, verejné alebo súkromné a pri ich návrhu sa uplatňuje princíp ukrývania implementačných detailov.



Obr. 3.3: Dáta a metódy v triedach Užívateľ a Skupina

3.2.5 5. vrstva: Návrh metód

5. vrstva návrhu obsahuje technické detaily implementácie metód. Návrh metód je obvykle ponechaný na vývojárov, ktorí budú metódu implementovať. Z návrhu priamo vyplývajú systémové požiadavky aplikácie na výkon, kapacity a hardvérové a softvérové vybavenie.

3.3 Nástroje

V tejto kapitole sme už zadefinovali návrh architektúry aplikácie a niektoré vlastnosti návrhu, na ktoré by vývojári pri tvorbe návrhu mali myslieť. Teraz uvedieme nástroje, ktoré môžu tvorbu (nielen) návrhov zjednodušiť. V tejto časti povieme niečo o prototypoch, Unified Modeling Language (UML) a návrhových vzoroch.

3.3.1 Prototypy

Prototypy vo všeobecnosti slúžia na odskúšanie nových nápadov. Účelom každého vytváraného prototypu je zodpovedanie niektorých konkrétnych otázok, preto pri jeho tvorbe môžeme mnohé faktory ignorovať. Tieto faktory sú nepodstatné pre

zodpovedanie otázok, ale môžu mať zásadný vplyv na výsledný produkt. Prototypy pomáhajú rozhodovať sa počas tvorby, analyzovať vplyvy rôznych faktorov či vizualizovať produkt pre ďalšie osoby. Tvorba prototypov nás síce stojí určitý čas a prostriedky, ale ich návratnosť v prípade výskytu chyby alebo chýb v produkte je veľmi rýchla. Z rovnakých dôvodov sa prototypy používajú aj pri vývoji webových aplikácií, či už ide o návrh architektúry alebo stavbu aplikácie.

„Prototypy si obvykle predstavujeme vo forme kódu, ale nemusí tomu vždy tak byť. Vynikajúcim nástrojom pre overovanie dynamických vecí, akými sú pracovné postupy alebo aplikačná logika, sú poznámkové štítky. Prototypy užívateľského rozhrania je možné kresliť na tabuľu alebo sa dá vytvoriť nefunkčný model nakreslený v kresliacom programe.“ [HT99]

Na prototypoch sa hodí testovať všetko, čo predstavuje riziko, čo nebolo doteraz odskúšané, všetko kritické (pre výslednú aplikáciu), neoverené, pokusné, pochybné či čokoľvek, s čím nie sme spokojní. Prototypy je možné použiť napríklad pri tvorbe architektúry (napríklad i užívateľského rozhrania), na odskúšanie nových funkcií a nástrojov alebo pri riešení problému výkonnosti.
„Prototypy nám poskytujú odpovede. Ich zmysel nespočíva vo vytváraní kódu, ale v získaných skúsenostiach.“ [HT99]

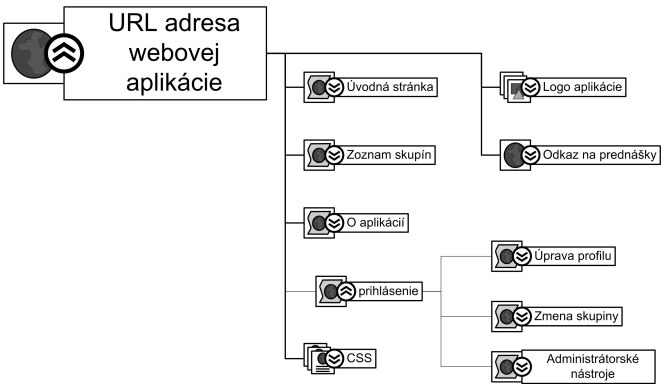
Andrew Hunt a David Thomas vo svojej knihe [HT99] hovoria o niekoľkých vlastnostiach detailov, ktoré pri tvorbe prototypov môžeme ignorovať:

- **Správnosť.** Je možné pracovať s jednoduchými dátami, ak nie sú pre prototyp dôležité.
- **Úplnosť.** Prototypy môžu mať značne obmedzenú funkčnosť (napríklad len pre vybrané vstupné dáta)
- **Robustnosť.** Niektoré časti prototypu ostanú nedokončené. Pre prototypy napríklad nemusí byť dôležitá kontrola chýb[6].
- **Dokumentácia.** Prototypy zvyčajne nevyžadujú rozsiahle komentáre a dokumentáciu.

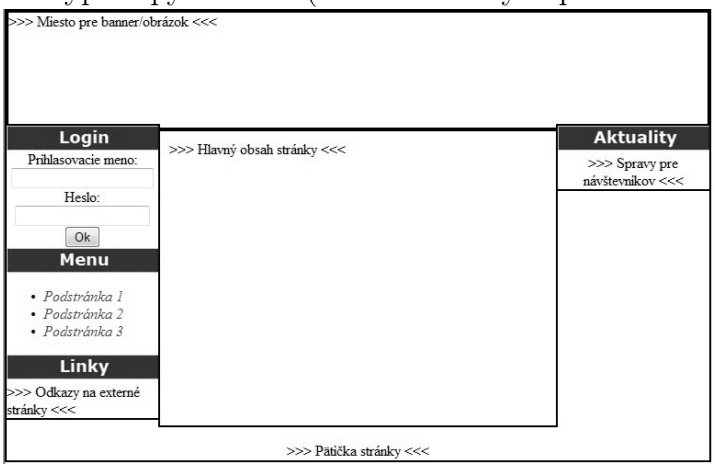
Prototypy nie sú súčasťou výslednej aplikácie, po použití by sa mali „zahodiť“ a finálny návrh (prípadne kód) by sme mali nanovo napísať avšak už s ohľadom na výsledky, ktoré nám prototypy poskytli. Vzhľadom na vlastnosti prototypov (ignorovanie niektorých faktorov, používanie nesprávnych alebo neúplných dát, atď.) nie je vhodné vytvorené prototypy ďalej používať pri tvorbe ďalších častí

[6]Pre výslednú odpoveď prototypu na otázky, ktoré ním riešime, môže byť napríklad bezpečnostná diera v podobe špecifických vstupných dát nepodstatná, preto nemá zmysel vstupy ošetrovať.

aplikácie.



Obr. 3.4: Prototyp mapy stránok (zoznam všetkých podstránok a odkazov)



Obr. 3.5: Prototyp návrhu rozmiestnenia prvkov užívateľského rozhrania

3.3.2 Unified Modeling Language (UML)

UML je štandardizovaný jazyk na návrh a vizualizáciu systémov a ich častí, na ktorého vzniku sa podielali Grady Booch, James Rumbaugh a Ivar Jacobson. „Jazyk UML sa skladá z mnohých grafických prvkov, ktoré sa dajú podľa pravidiel jazyka vzájomne kombinovať do podoby diagramov. UML definuje 13 diagramov rozdelených do 3 skupín:

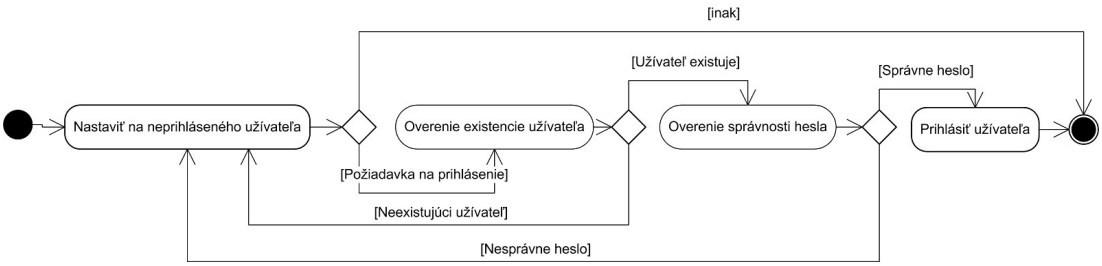
- Diagramy správania (use-case diagram, state chart diagram, activity diagram)
- Štrukturálne diagramy (class diagram, object diagram, component diagram, composite structure diagram, package diagram, deployment diagram)

- Interakčné diagramy (collaboration diagram, communication diagram, sequence diagram, interaction overview)“ [Pec07]

Štandard UML definuje Object Managment Group (OMG, http://www.uml.org), kde je možné nájsť bližšie informácie o UML, tutoriály ako aj špecifikáciu štandardu[7].

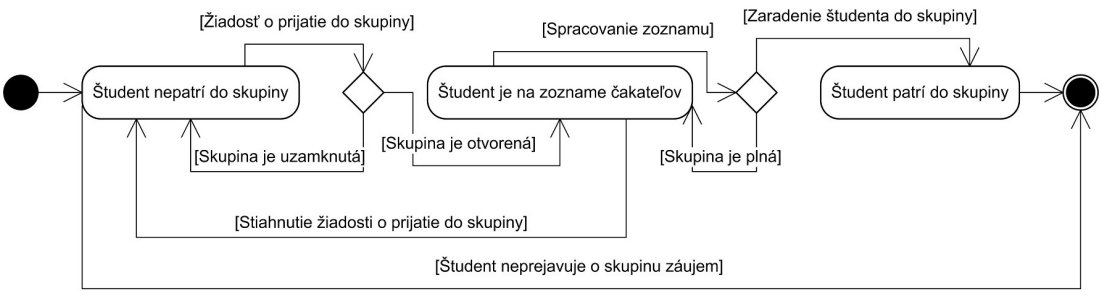
Ukážky použitia UML teraz demonštrujeme na diagramoch správania, v kapitole Databázy demonštrujeme použitie Class diagramu na návrh databázy, použitie zvyšných diagramov ponechávame na čitateľa.

Activity diagram



Obr. 3.6: Activity diagram reprezentujúci aktivity počas pokusu o prihlásenie užívateľa

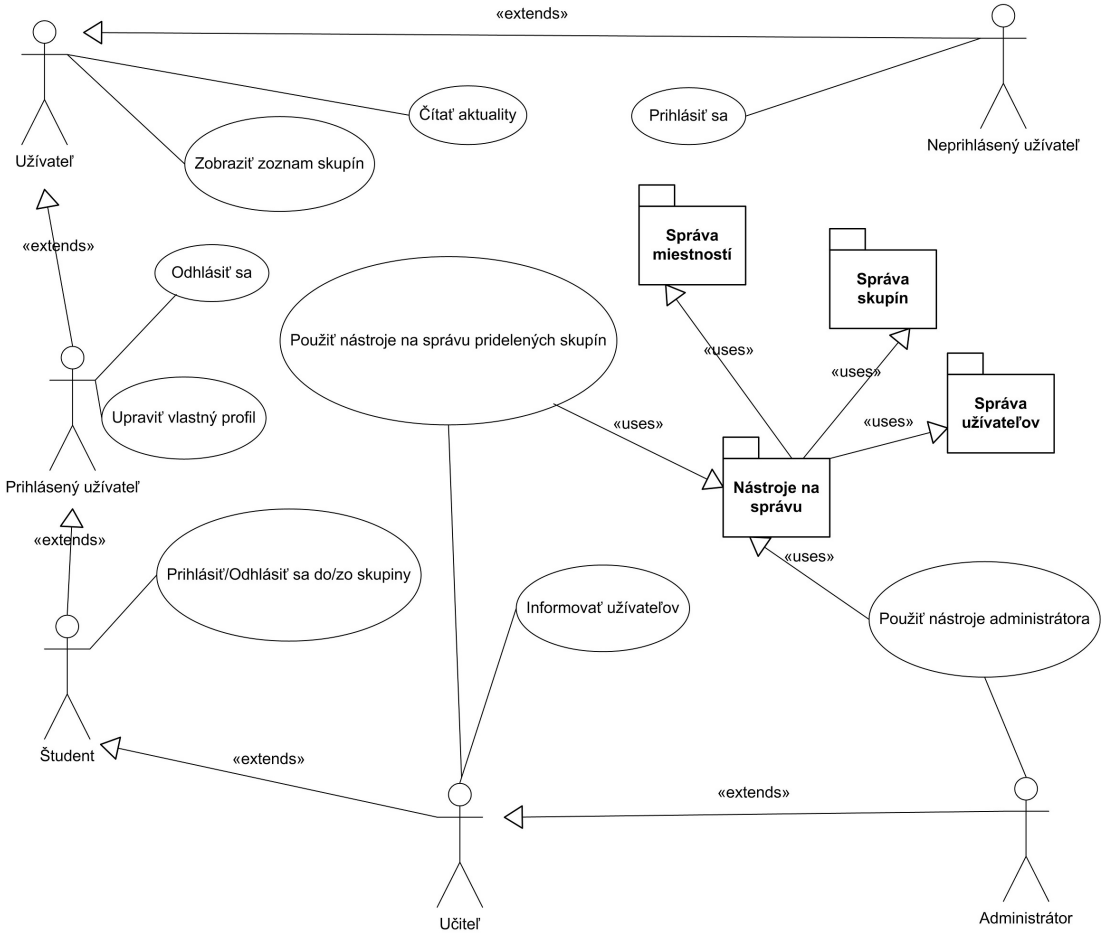
State chart diagram



Obr. 3.7: State chart diagram reprezentujúci stav študenta počas pokusu o prihlásenie do skupiny

[7] http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML

Use case diagram



Obr. 3.8: Use case diagram reprezentujúci užívateľov a ich možnosti práce s aplikáciou

3.3.3 Návrhové vzory

Návrhové vzory sú všeobecné riešenia veľmi často sa vyskytujúcich problémov, používané sú najmä v objektovo orientovanom programovaní, avšak niektoré z nich sú v určitej miere použiteľné aj v procedurálnom programovaní. Sú to vlastne akési vopred pripravené šablóny, pomocou návrhových vzorov je tak možné rýchlejšie riešiť mnohé úlohy návrhu architektúry. Návrhových vzorov je niekoľko desiatok a delia sa do troch základných skupín — vytvárajúce vzory (creational patterns, týkajú sa vytvárania objektov), štrukturálne vzory (structural patterns, usporiadanie tried, sprehľadnenie systému) a vzory správania (behavioral patterns, vystupovanie a komunikácia medzi objektami v systéme). V tejto časti uvedieme niektoré z návrhových vzorov, ktoré považujeme za najpoužívanejšie

vo webových aplikáciách, a na príkladoch demonštrujeme ich použitie. Ďalšie návrhové vzory popisuje Rudolf Pecinovský vo svojej knihe [Pec07].

Šablónová metóda (Template method)

Šablónová metóda patrí medzi návrhové vzory správania. „Definuje kostru algoritmu v operácii delegujúc niektoré kroky na podtriedy. Šablónová metóda umožnuje podtriedam pozmeniť niektoré kroky algoritmu bez toho, aby zmenila jeho štruktúru.“ [Pec07] Patrí k najpoužívanejším návrhovým vzorom [Pec07] a umožňuje vyhýbať sa duplicitným metódam, ktoré sa líšia iba v niektorých detailoch, pričom základný algoritmus ostáva rovnaký.

Fasáda (Facade)

Fasáda patrí medzi štrukturálne návrhové vzory. „Poskytuje jednotné rozhranie k množine rozhraní v subsystéme. Definuje rozhranie vyššej úrovne, ktoré zjednoduší používanie subsystému.“ [Pec07] Fasádu je možné použiť pri integrácii komponentov tretích strán do systému, ak potrebujeme iba časť poskytovanej funkcionality. Pomocou tohto návrhového vzoru je možné vytvoriť jedinú funkciu, ktorá implementuje potrebnú činnosť len pomocou postupnosti volaní vybraných funkcií komponentu. Ďalším dôvodom pre použitie Fasády je zníženie počtu parametrov niektorej funkcie, pričom zvyšné parametre sú v rámci rozhrania pevne definované.

```
function db_connect($host,$dbname,$user,$password,$schema)
  {
  $dbconn = pg_connect("host=$host dbname=$dbname user=$user password=$password");
  if ($schema != "")
    {
    pg_query("SET search_path TO $schema");
    }
  return $dbconn;
  }
```

Obr. 3.9: Použitie návrhového vzoru Fasáda pre vytvorenie funkcie, ktorá vytvorí databázové spojenie a nastaví schému, ak je to potrebné. Pri vytváraní databázového spojenia tak voláme už iba jednu funkciu namiesto dvoch

Most (Bridge)

Ďalším zo štrukturálnych návrhových vzorov je Most. „Vzor zavedením rozhrania oddeľuje abstrakciu sprostredkovávajúcu nejakú službu od implementácie tejto služby, takže je možné obidve meniť nezávisle na sebe.“ [Pec07] Pri tvorbe webových stránok má Most dôležité postavenie, umožňuje napríklad jednoduchú integráciu prototypov do systému pre potreby testovania a následné jednoduché nahradenie prototypov za finálny kód. Pomocou návrhového vzoru most je možné

tiež pripraviť aplikáciu na použitie s rôznymi databázovými systémami podľa potrieb konkrétneho zákazníka.

```
$vysledok = db_query("select * from users where user_id='".db_safe_string($_SESSION['user_id'])."'");
$zaznam = db_fetch_array($vysledok);
```

db_functions_mysql.php

```
function db_query($query)
  {
  return @mysql_query($query);
  }

function db_fetch_array($query_result)
  {
  return @mysql_fetch_array($query_result);
  }

function db_num_rows($query_result)
  {
  return @mysql_num_rows($query_result);
  }

function db_error()
  {
  return mysql_error();
  }

function db_safe_string($string)
  {
  return mysql_real_escape_string($string);
  }
```

db_functions_postgresql.php

```
function db_query($query)
  {
  return @pg_query($query);
  }

function db_fetch_array($query_result)
  {
  return @pg_fetch_array($query_result);
  }

function db_num_rows($query_result)
  {
  return @pg_num_rows($query_result);
  }

function db_error()
  {
  return pg_last_error();
  }

function db_safe_string($string)
  {
  return pg_escape_string($string);
  }
```

Obr. 3.10: Použitie návrhového vzoru Most pre vytvorenie univerzálnych funkcií pre prácu s rôznymi databázami

Veľmi podobným návrhovým vzorom k vzoru Most je vzor Stratégia (Strategy, vzor pre správanie), ktorý navyše umožňuje systému na základe nastavených parametrov vybrať správnu implementáciu služby.

Kapitola 4

Tímová práca

V tejto kapitole sa budeme venovať vybraným problémom riadenia vývojárskych tímov, ktoré demonštrujeme na príkladoch aplikácie pre podporu organizácie cvičení.

4.1 Hierarchia

Tvorbu menších webových aplikácií[1] zvyčajne zvládne jeden vývojár, ktorý si sám pripraví návrh architektúry podľa požiadaviek, vytvorí si harmonogram prác, implementuje svoj návrh podľa stanoveného harmonogramu, vykoná testy a následne k aplikácii spíše dokumentáciu. Vývojár komunikuje iba so zadávateľom projektu a koncovými užívateľmi. V každom okamžiku má presnú predstavu o stave celého projektu, ktorá mu umožňuje uskutočniť zmeny v projekte za účelom zefektívnenia procesu tvorby aplikácie. V rámci vývojárskeho tímu neprebieha žiadna komunikácia, vďaka čomu je možné všetok čas venovať tvorbe aplikácie. Pre malé projekty sa zdá byť jednočlenný tím ako rozumná voľba. S veľkosťou projektu však rastie aj počet hodín, ktoré vývojár strávi tvorbou aplikácie.

Pri projektoch väčších rozmerov[2] tak často nastáva situácia, pri ktorej nie je v silách jedného vývojára dokončiť projekt v stanovenom termíne, preto je nevyhnutné zapojiť do procesu viac vývojárov. Na prvý pohľad sa môže zdať, že s pribúdajúcim počtom vývojárov nepriamo úmerne klesá čas potrebný na dokončenie projektu. Táto úvaha by však bola pravdivá len za predpokladu, že všetci vývojári rozmýšľajú úplne rovnako (tzv. kolektívne vedomie) alebo časti, na ktorých pracujú vývojári, sú úplne nezávislé na zvyšku aplikácie.

V prípade 2 a viacčlenných tímov do procesu vývoja aplikácie vstupuje aj potreba komunikácie pri prerozdeľovaní prác, objasňovaní cieľov, kontrolách stavu

[1] Napr. školské projekty, domovské stránky, jednoduché e-shopy a ďalšie.

[2] Čo do počtu riadkov zdrojového kódu a zložitosti (použitých technológií).

projektu a pri riešení problémov a chýb, na ktoré jednotliví vývojári v priebehu vývoja aplikácie natrafia. V prípade rovnocenného postavenia všetkých vývojárov komunikuje každý vývojár s každým vývojárom. Počet komunikačných kanálov teda môžeme vyjadriť vzťahom:

$$Počet\ komunikačných\ kanálov = \frac{n(n-1)}{2}$$

Napríklad v 3-člennom tíme sú teda 3 komunikačné kanály, v 5-člennom tíme je ich už 10 a pri 8-člennom tíme je ich až 28. Diskusia o riešení nejakého problému alebo rozhodovanie o smerovaní projektu vo väčších tímoch bude v takomto prípade značne problematická, pretože vyšší počet komunikačných kanálov má za následok vyšší čas strávený komunikáciou namiesto vývoja a zvýšenie rizika vzniku nedorozumení a chýb. Ukazuje sa teda, že je potrebné zaviesť určitú formu organizácie v rámci vývojárskeho tímu.

„Základná rola vedúcich je vždy rovnaká. Spočíva v pomoci skupine splniť spoločnú úlohu, udržať ju ako celok a zaistiť, aby každý člen prispel podľa svojich najlepších možností - stimulovať rozvoj každého člena.“

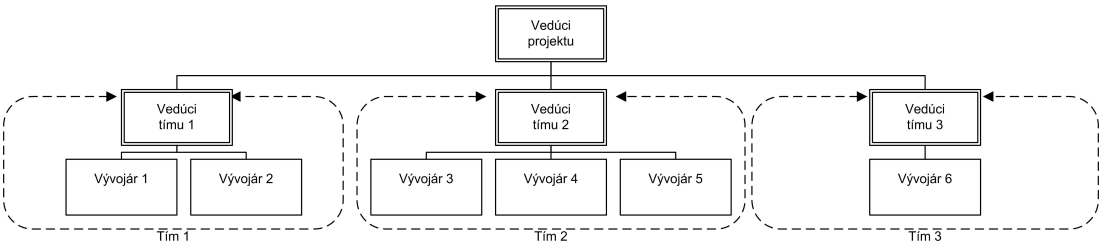
Gilbert Adair

Vedúci tímu je vývojár, ktorý je zodpovedný za riadenie tímu a úspešné ukončenie projektu. Hlavnými úlohami vedúceho tímu sú

- rozhodovanie o smerovaní projektu
- komunikácia s členmi tímu a ich motivácia[3]
- prerozdelenie práce vývojárom a tvorba časového harmonogramu
- riešenie problémov a konfliktov v rámci projektu a tímu
- udržiavanie prehľadu o aktuálnom stave projektu
- kontrola postupu prác
- a komunikácia s ďalšími účastníkmi projektu.

Ak komunikáciu vnútri vývojárskeho tímu obmedzíme len na komunikáciu medzi vedúcim tímu a vývojárom, počet potrebných komunikačných kanálov sa zníži na hodnotu 2(n-1), kde n je veľkosť tímu. V prípade 8-členného tímu je to teda pokles o 14 kanálov, teda o 50%. Nevýhodou takéhoto riešenia je potreba výberu kvalitného vedúceho tímu, ktorí má dobré vodcovské a komunikačné schopnosti, dokonale ovláda problematiku a zvláda vyššiu záťaž.

[3]Z mnohých motivačných teórií vyberáme: Maslowova hierarchia potrieb, Herzbergova teória motivačnej hygieny, McClellandova teória získaných potrieb, McGregorova teória X a teória Y, teória vplyvu a sily podľa Thamhaina a Wilemona a ďalšie.



Obr. 4.1: Príklad rozdelenia vývojárskeho tímu

4.1.1 Divide et impera — Rozdeľuj a panuj

V prípade väčších tímov sa ukazuje, že je dobré jeden veľký tím rozdeliť na niekoľko menších skupín, ktorým budú pridelené vybrané úlohy a vedúci skupiny, ktorý bude podliehať vedúcemu tímu a bude mať v rámci skupiny rovnaké úlohy ako vedúci tímu. Vytvorením takejto hierarchickej štruktúry dosiahneme ďalšie zníženie potrebných komunikačných kanálov a zefektívnenie práce tímu.
„Čím väčšia je daná skupina, tým viac sa prejavujú rôzne problémy s riadením. Pri zlej komunikácii sa exponenciálne zvyšuje riziko, že urobíte doslova fatálnu chybu. Projekt rozsiahleho merítka má množstvo „pohyblivých súčiastok“, takže sa môže podstatne ľahšie „pokaziť“. A komunikácia je mazivo, ktoré tento stroj udržuje v chode. Zaoberať sa atmosférou nedôvery a vyriešiť ju je o mnoho ľahšie v skupine piatich ľudí než v obrovskom tíme zloženom z 500 pracovníkov.“ [Hil96]

4.2 Rozpis prác

Keď už vieme, ako vyzerá organizácia členov nášho tímu, je čas, aby sme si povedali niečo o rozpise prác medzi jednotlivých vývojárov. „Štruktúra rozpisu prác, nazývaná tiež štruktúra rozkladu prác alebo štrukturovaná dekompozícia práce, je zoskupenie prác v projekte, ktoré je orientované na ucelené časti diela a ktoré definuje celkový rozsah projektu. Štruktúra rozpisu prác je základom pre celé plánovanie a riadenie časových plánov, nákladov, zdrojov a zmien v projekte.“ [Sch05] Medzi práce v projekte zaraďujeme návrh architektúry, stavbu aplikácie, testovanie a tvorbu dokumentácie. Je teda zrejmé, že rozpis prác vzniká postupne, nakoľko rozsah stavby vieme určiť až po vytvorení návrhu architektúry.

Aktuality	Databázový MicroFramework	Beta testovanie
Autentifikácia	Prioritná fronta čakateľov na skupinu	Dokumentácia
Dizajn	Profily užívateľov	E-mailová notifikácia
Návrh databázy	Správa aplikácie pre administrátorov	Zoznam skupín
Návrh systému	Správa skupín pre učiteľov	

Obr. 4.2: Zoznam úloh v projekte tvorby systému pre podporu organizácie cvičení z predmetu Úvod do databázových systémov

4.3 Časový harmonogram

Časový harmonogram je zoznam úloh z rozpisu prác doplnený o závislosti medzi úlohami a predpokladané dátumy začatia a ukončenia úloh.

4.3.1 Sieťový diagram

Najvhodnejšou technikou pre zoradenie úloh a vytvorenie časového harmonogramu je tvorba sieťového diagramu. [Sch05] Sieťový diagram (alebo sieťový graf) je acyklický orientovaný graf s ohodnotenými hranami, schématicky znázorňuje úlohy v projekte a logické vzťahy medzi nimi. Vrcholy diagramu tvoria začiatky a konce úloh, hrany predstavujú vykonávanie úlohy a ich ohodnotenie reprezentuje dĺžku času potrebnú na ich vykonanie. Prvý vrchol označuje začiatok projektu, posledný vrchol koniec projektu. Sieťový diagram je možné vytvoriť nasledujúcim postupom:

1. Do prázdneho grafu vložíme vrchol označujúci začiatok projektu. Vytvoríme zoznam všetkých úloh z rozpisu prác.

2. Zo zoznamu vyberieme všetky úlohy, ktoré sú od iných úloh nezávislé. Pre každú takúto úlohu vytvoríme novú ohodnotenú orientovanú hranu grafu s cenou rovnou odhadovanej dĺžke času potrebného na jej splnenie, počiatočný vrchol je vrchol označujúci začiatok projektu a koncový vrchol je nový vrchol grafu. Po zaradení hrany do grafu odstránime úlohu zo zoznamu.

3. Pre každý takto vytvorený nový vrchol vyberieme zo zoznamu všetky úlohy, ktorých realizácia vyžaduje ukončenie aspoň jednej z úloh už zaradených do grafu. Pre každú takúto úlohu vytvoríme novú ohodnotenú orientovanú hranu grafu s cenou rovnou odhadovanej dĺžke času[4] potrebného na jej splnenie. Počiatočný vrchol hrany je koncový vrchol najkratšej cesty v grafe vychádzajúcej z vrchola reprezentujúceho začiatok projektu a obsahujúcej všetky vyžadované úlohy reprezentované vo forme hrán. Ak takýto vrchol v grafe neexistuje, počiatočným bodom je nový bod grafu, ktorý je orientovanými hranami s ohodnotením 0 spojený s koncovými bodmi všetkých ciest, ktoré obsahujú vyžadované úlohy. Koncový vrchol hrany je nový vrchol grafu. Po zaradení hrany do grafu odstránime úlohu zo zoznamu.

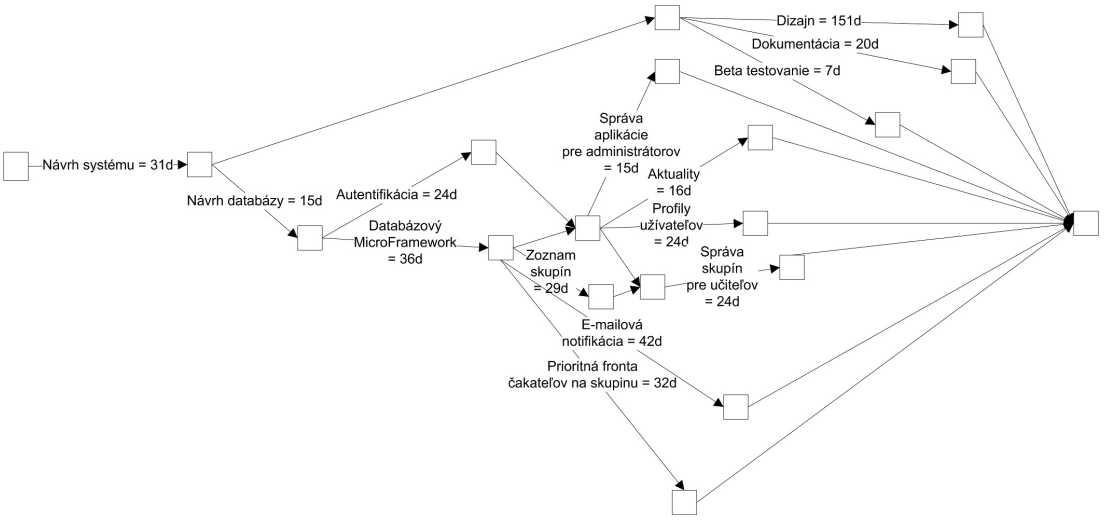
4. Opakujeme bod 3 až do vyprázdnenia zoznamu.

5. Ak je zoznam úloh prázdny, vytvoríme nový vrchol grafu označujúci koniec projektu a zo všetkých ostatných vrcholov grafu, z ktorých nevedie žiadna ďalšia hrana, vedieme nové hrany s ohodnotením 0 do vrcholu označujúceho koniec projektu.

[4] Je vhodné do odhadu započítať i časovú rezervu pre prípad výskytu mimoriadnej situácie.

Uvedeným postupom sme vytvorili sieťový graf, ktorý reprezentuje časový harmonogram nášho projektu. Pokiaľ je to možné, je vhodné graf prekresliť na izomorfný planárny graf[5].

Z takto vytvoreného sieťového grafu vieme zistiť odhad dĺžky projektu, jednotlivých jeho častí a množinu úloh, ktoré môžeme vykonávať paralelne. Odhadovaná dĺžka celého projektu je dĺžka najdlhšej cesty z vrchola začiatku do vrchola konca projektu.



Obr. 4.3: Sieťový diagram úloh

4.3.2 Ganttov diagram

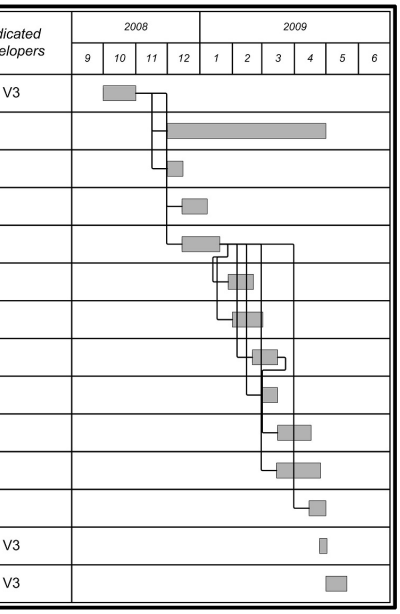
„Ganttov diagram je štandardný formát pre grafické zachytenie informácií o časovom pláne projektu, v ktorom sú uvedené jednotlivé úlohy projektu a im odpovedajúci dátumom zahájenia a ukončenia v kalendárovom formáte. Niekedy je označovaný ako pruhový diagram, pretože úlohy sú v ňom označené ako vodorovné pruhy, ktoré vedú vždy od dátumu zahájenia k dokončeniu úlohy.“ [Sch05]

Ganttov diagram je možné ľahko vytvoriť zo sieťového diagramu. Ak jednotlivým úlohám priradíme vývojárov alebo skupiny, ktoré nesú zodpovednosť za splnenie danej úlohy, a vykonáme prípadné úpravy v diagrame[6], získame možnosť efektívneho naplánovania projektu, využitia vývojárskeho tímu a kontroly plnenia plánu.

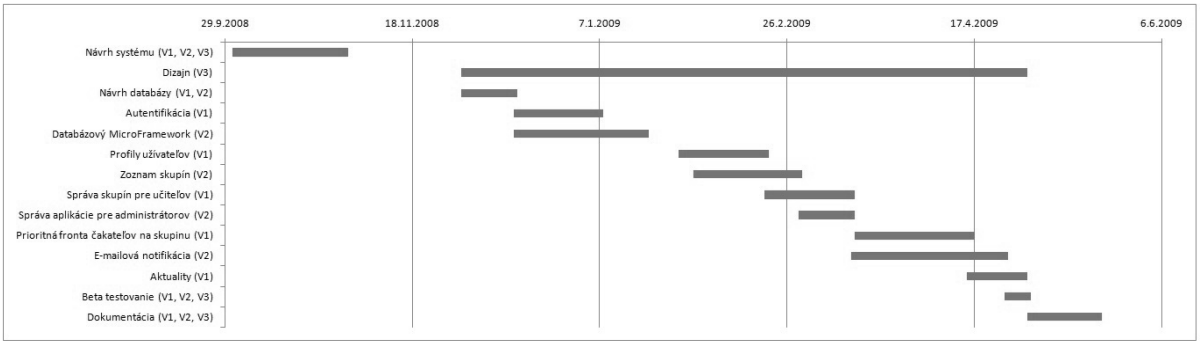
[5]Graf s rovnakou množinou vrcholov a hrán, v ktorom navyše platí, že žiadne dve rôzne hrany nemajú spoločný bod s výnimkou krajných vrcholov.

[6]Je možné, že jeden vývojár sa nemôže v tom istom čase venovať riešeniu viac ako jednej úlohy, preto je v prípade obmedzenej veľkosti tímu nevyhnutné niektoré úlohy presunúť na iný termín samozrejme pri zachovaní všetkých závislostí.

ID	Task Name	Start	Finish	Duration	Dedicated Developers
1	Návrh systému	1. 10. 2008	31. 10. 2008	31d	V1, V2, V3
2	Dizajn	1. 12. 2008	30. 4. 2009	151d	V3
3	Návrh databázy	1. 12. 2008	15. 12. 2008	15d	V1, V2
4	Autentifikácia	15. 12. 2008	7. 1. 2009	24d	V1
5	Databázový MicroFramework	15. 12. 2008	19. 1. 2009	36d	V2
6	Profily užívateľov	28. 1. 2009	20. 2. 2009	24d	V1
7	Zoznam skupín	1. 2. 2009	1. 3. 2009	29d	V2
8	Správa skupín pre učiteľov	20. 2. 2009	15. 3. 2009	24d	V1
9	Správa aplikácie pre administrátorov	1. 3. 2009	15. 3. 2009	15d	V2
10	Prioritná fronta čakateľov na skupinu	16. 3. 2009	16. 4. 2009	32d	V1
11	E-mailová notifikácia	15. 3. 2009	25. 4. 2009	42d	V2
12	Aktuality	15. 4. 2009	30. 4. 2009	16d	V1
13	Beta testovanie	25. 4. 2009	1. 5. 2009	7d	V1, V2, V3
14	Dokumentácia	1. 5. 2009	20. 5. 2009	20d	V1, V2, V3



Obr. 4.4: Ganttov diagram vytvorený špecializovanej aplikácie na tvorbu diagramov



Obr. 4.5: Ganttov diagram vytvorený ako graf v tabuľkovom procesore. Ak nepožadujeme dodatočné funkcie, tabuľkový procesor, ktorý je bežnou súčasťou kancelárskych balíkov, sa ukazuje ako vhodná alternatíva

Kapitola 5

Dizajn webových stránok

Popri funkcionalite, ktorú aplikácia užívateľom poskytuje, je dôležitou súčasťou webovej aplikácie užívateľské rozhranie vo forme webových stránok, ktoré tvorí jedinú možnosť komunikácie užívateľa s webovou aplikáciou. Dizajn webových stránok je preto dôležitou súčasťou tvorby webových aplikácií, ktorej je venovaná táto kapitola. Zameriame sa na dva dôležité aspekty dizajnu webových stránok — použiteľnosť a prístupnosť.

5.1 Použiteľnosť

„Použiteľnosť je kvalitatívny atribút, ktorý hodnotí ako jednoduché je použitie užívateľského rozhrania. Použiteľnosť je definovaná piatimi zložkami kvality:

- **Prvé použitie** určuje ako jednoduché je pre užívateľov vykonať základné úlohy pri prvom stretnutí sa s dizajnom.
- **Efektivita** určuje ako rýchlo dokážu užívatelia vykonávať úlohy, keď sa oboznámia s dizajnom.
- **Zapamätateľnosť** určuje ako jednoducho dokážu užívatelia nadobudnúť zručnosť s dizajnom, keď sa k nemu vrátia po dlhšej dobe.
- **Chybovosť** určuje koľko chýb užívatelia urobia, ich váhu a ako ľahko dokážu chyby napraviť.
- **Spokojnosť** určuje príjemnosť práce s dizajnom.“ [Nie]

Univerzálne riešenie použiteľnosti vzhľadom na závislosť od skupiny konečných užívateľov neexistuje, avšak Jakob Nielsen na svojej stránke [Nie] uvádza niekoľko riešení pre vybrané prípady. Vo všeobecnosti platí, že použiteľnosť užívateľského rozhrania vytvoreného na základe požiadaviek a hodnotení prototypov rozhrania je potrebné testovať na reprezentatívnej vzorke jeho budúcich užívateľov, ktorí s ním vykonajú požadované úlohy a následne sú požiadaní o zhodnotenie rozhrania.

5.2 Prístupnosť

Prístupnosť dizajnu rieši otázky dostupnosti webových stránok pre hendikepovaných užívateľov, mobilné zariadenia a spracovanie počítačmi. S cieľom zlepšiť prístupnosť webových aplikácií vytvorilo konzorcium W3C v spolupráci s americkým Bielym domom iniciatívu Web Accessibility Initiative (WAI), ktorá vytvorila metodiku pre tvorbu prístupných webstránok Web Content Accessibility Guidelines (WCAG). Aktuálne odporúčaná verzia WCAG 2.0[1] uvádza tri úrovne prístupnosti v závislosti od splnenia jednotlivých kritérií: A, AA, AAA.

Kritéria sú rozdelené do štyroch skupín podľa princípov:

- **Zrozumiteľnosť.** Informácie a časti užívateľského rozhrania musia byť prezentované užívateľom zrozumiteľným spôsobom. Do tejto skupiny sa zaraďujú alternatívne popisy k prvkom rozhrania, práca s textom a s farbami či alternatívne formy prezentácie multimédií (videa a zvuku).

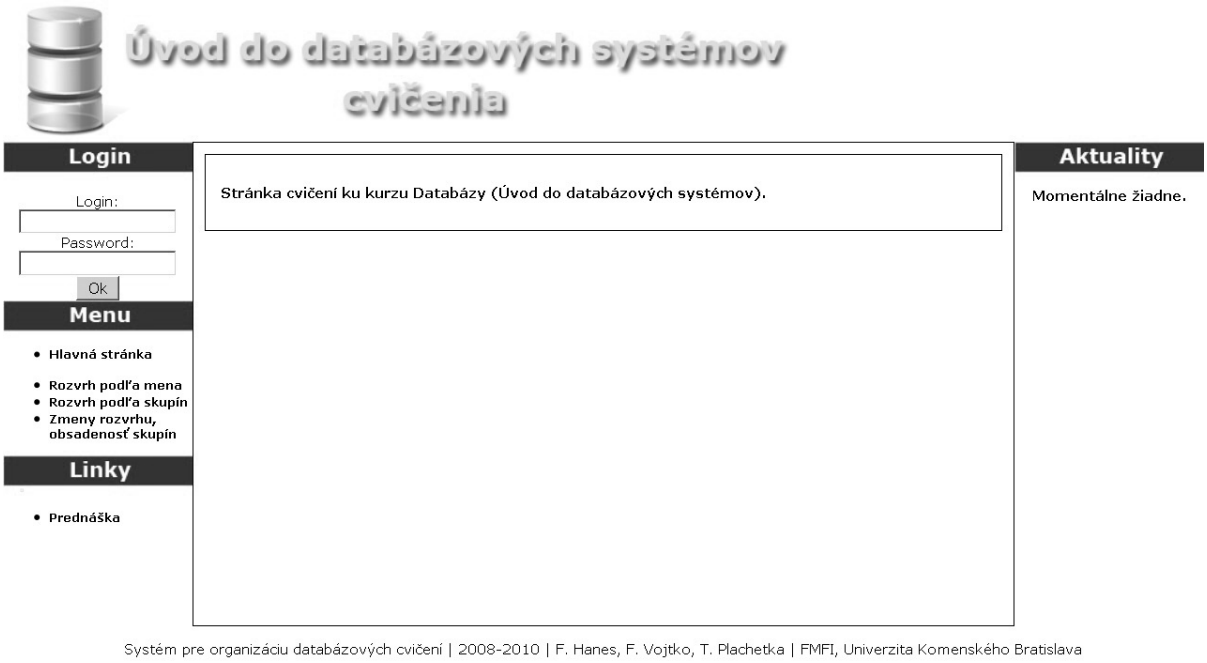
- **Funkčnosť.** Časti rozhrania a navigácie musia byť funkčné, dostupné a použiteľné. Patrí sem ovládanie klienta iba za pomoci klávesnice, riešenie časových obmedzení, navigácie a predchádzanie nevôli užívateľov.

- **Porozumenie.** Užívateľ musí rozumieť informáciám a funkciám užívateľského rozhrania. V tejto skupine sa nachádzajú kritériá čitateľnosti, predvídateľnosti a pomoc pri zadávaní vstupných údajov.

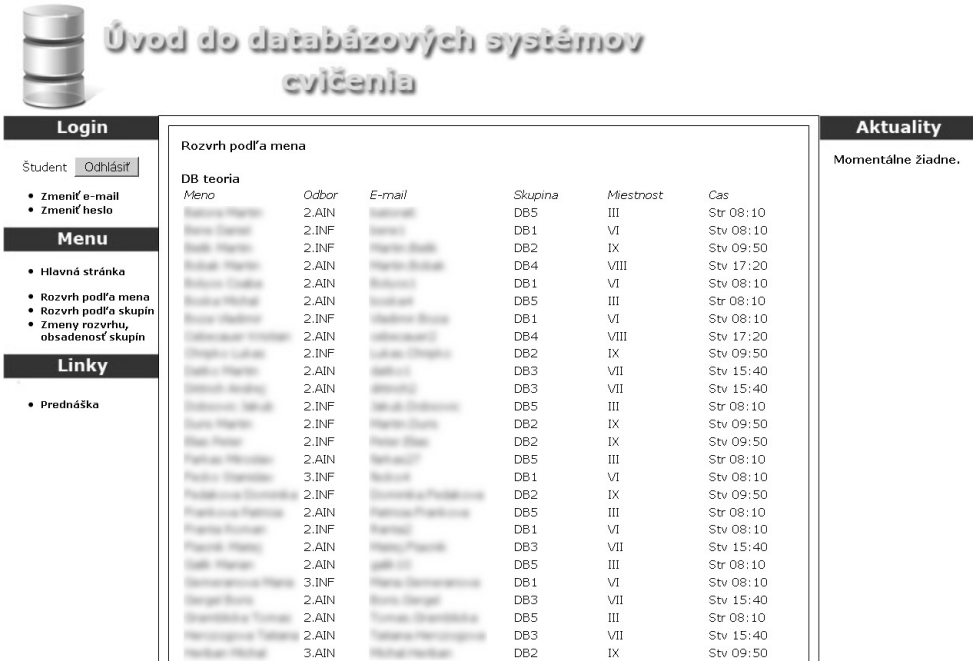
- **Robustnosť.** Obsah musí byť dostatočne robustný, aby ho bolo možné spoľahlivo interpretovať širokou skupinou rôznych klientov. Patrí sem kompatibilita so štandardami.

Na tvorbu webových aplikácií a dosiahnutie prístupnosti webových aplikácií širokej skupine užívateľov sa v súčasnosti používajú jazyky HTML[2,3], XHTML[4,5] a CSS[6], ktorých (aspoň čiastočná) podpora je zahrnutá do všetkých moderných webových prehliadačov. Obsah webovej stránky je oddelený od prezentácie — HTML a XHTML slúžia na vytvorenie obsahu, základnej štruktúry webovej stránky, CSS formátuje obsah stránky a vytvára tak výslednú prezentáciu. Vďaka ich masívnemu rozšíreniu a podpore existujú aj validátory týchto jazykov[7,8].

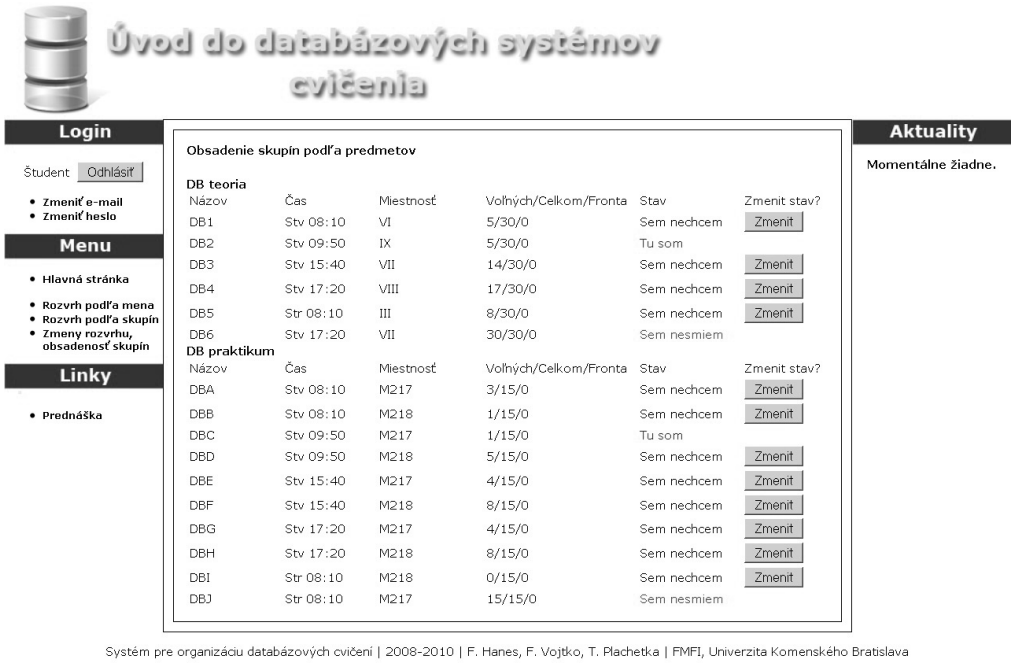
[1] http://www.w3.org/TR/WCAG20
[2] HTML 4.01: http://www.w3.org/TR/html401
[3] HTML 5: http://www.w3.org/TR/html5
[4] XHTML 1.0: http://www.w3.org/TR/xhtml1
[5] XHTML 1.1: http://www.w3.org/TR/2001/REC-xhtml11-20010531
[6] CSS 2.1: http://www.w3.org/TR/CSS21
[7] http://validator.w3.org
[8] http://jigsaw.w3.org/css-validator



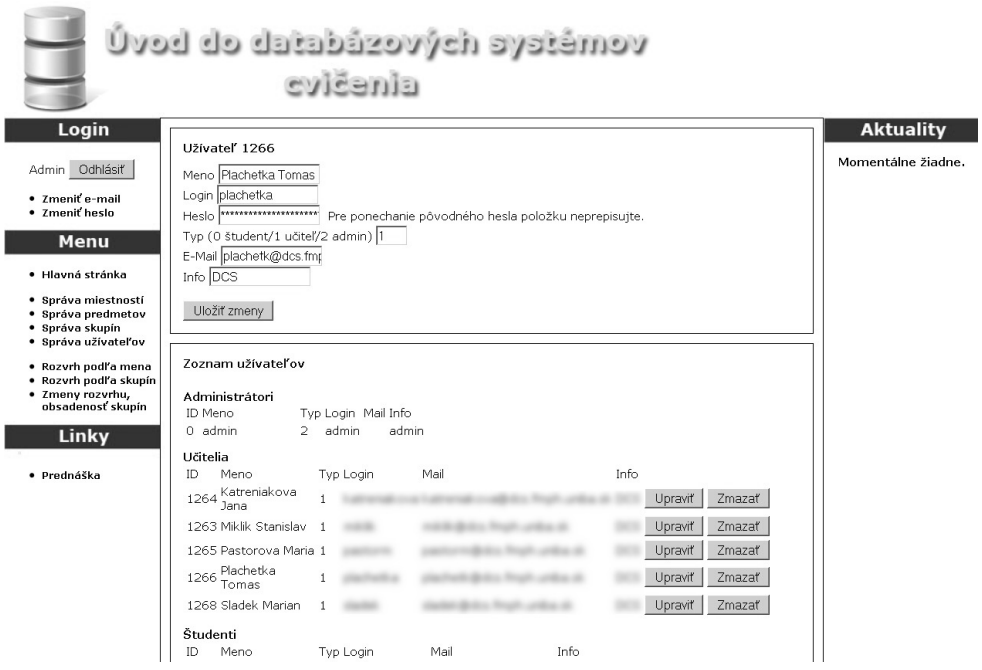
Obr. 5.1: Základná stránka systému pre podporu organizácie cvičení z predmetu z Úvodu do databázových cvičení



Obr. 5.2: Zoznam užívateľov systému s priradenými skupinami a časmi konania cvičení. Mená užívateľov a ich e-mailové adresy sú z dôvodu ochrany osobných údajov nečitateľné

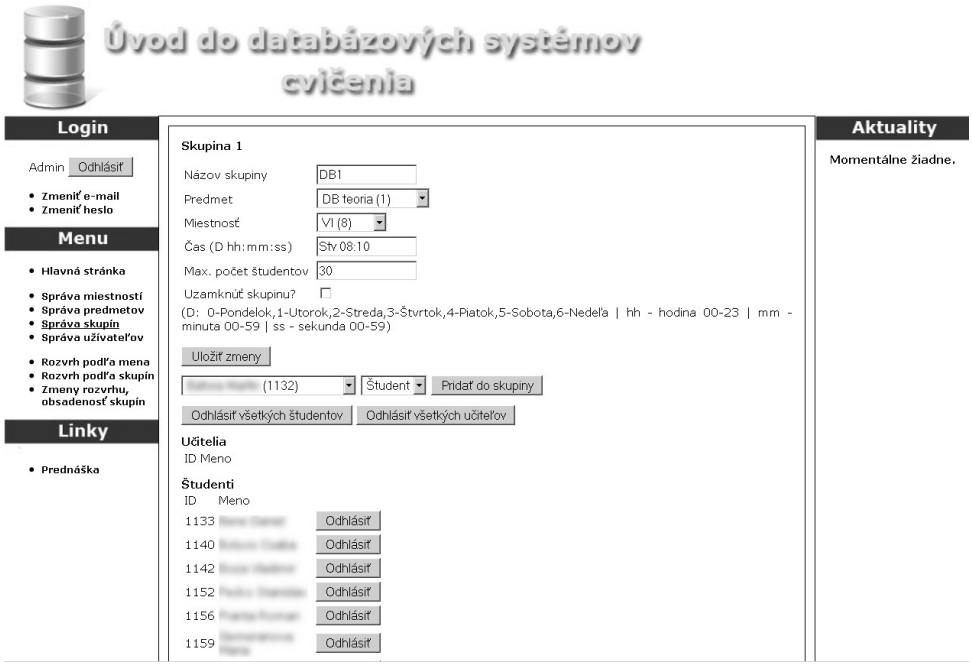


Obr. 5.3: Zoznam všetkých skupín s možnosťou prihlásenia a odhlásenia sa užívateľov do/zo skupín



Obr. 5.4: Nástroj pre administrátora na správu užívateľov

Obr. 5.5: Nástroj pre administrátora na správu skupín



Obr. 5.6: Nástroj pre administrátora na správu užívateľov — úprava vybranej skupiny

Kapitola 6

Webové aplikácie a bezpečnosť

Jedným z najdôležitejších problémov, s ktorými sa potýka (nielen) tvorba webovej aplikácie, je riešenie otázky bezpečnosti. Z hľadiska cieľa útoku môžeme riešenia bezpečnostných incidentov rozdeliť do niekoľkých skupín: ochrana koncových užívateľov, ochrana dát a ochrana servera. Útočník sa môže snažiť zneužiť chyby v klientskom prehliadači, v nastaveniach servera a/alebo vo webovej aplikácii. V tejto kapitole sa budeme venovať vybraným chybám v bezpečnosti webových aplikácií, ktorých sa vývojár môže dopustiť pri tvorbe aplikácie[1]. Spolu s popisom bezpečnostných rizík uvedieme spôsob ich ošetrenia a demonštrujeme ho na našej webovej aplikácii.

6.1 Najznámejšie typy útokov

6.1.1 Code injection

Útoky triedy code injection (vkladanie kódu) zneužívajú nedostatočnú kontrolu vstupných dát, ktoré sú najčastejšie prenášané v podobe znakových reťazcov a interpretujú sa niektorým subsystémom ako príkazy jazyka akceptovaného daným subsystémom. V takýchto jazykoch slúžia vybrané znaky ako riadiace znaky subsystému. V prípade nedostatočnej kontroly vstupných dát teda útočník na prienik do subsystému použije špeciálne pripravený znakový reťazec. Najčastejšie využívanými subsystémami a teda aj najčastejšími cieľmi útokov triedy code injection sú SQL databázové systémy a programy typu shell (príkazový riadok), ktoré popíšeme podrobnejšie.

Útokom triedy code injection je možné sa vo všeobecnosti vyhnúť dôslednou kontrolou vstupných dát, pri ktorej budú riadiace znaky subsystémov zo vstupných dát odstránené alebo pozmenené tak, aby ich subsystém ďalej neinterpre-

[1] Prevenciu a opravu chýb v klientských prehliadačoch a v konfigurácii servera ponechávame na čitateľa.

toval.

SQL Injection

SQL injection je názov pre útok triedy code injection zameraný na SQL databázový systém. Útočník využíva skutočnosť, že SQL príkaz je často na strane servera konštruovaný z predpripravenej šablóny, do ktorej sú na určených miestach vložené obsahy textových premenných, ktoré reprezentujú požiadavku zo strany užívateľa (napríklad informácie z formulárov). Tieto miesta v šablónach môžu byť ohraničené úvodzovkami alebo apostrofmi.

```
db_query("UPDATE user_profiles
          SET user_mail='$user_mail'
          WHERE user_id='$user'");
```

Obr. 6.1: Jeden z možných spôsobov vytvorenia a používania predpripravenej šablóny

Na prevzatie kontroly nad databázovým systémom je potrebné, aby útočník poznal alebo uhádol štruktúru takéhoto dotazu a aplikácii odoslal ako vstupné dáta reťazec začínajúci správnym znakom, za ktorým nasleduje ľubovoľný príkaz zo strany útočníka a prípadné odstránenie zvyšku šablóny za pomoci riadiacich znakov označujúcich začiatok komentára.

```
vstupný reťazec:
jozko@utocnik.sk' WHERE user_id=1 --

po dosadení do šablóny:
db_query("UPDATE user_profiles
          SET user_mail='jozko@utocnik.sk' WHERE user_id=1 --'
          WHERE user_id='$user'");
```

Obr. 6.2: Jeden z možných útokov, ktorého výsledkom je zmena e-mailovej adresy ľubovoľnému užívateľovi (v tomto prípade administrátorovi)

Ochrana proti SQL injection útokom spočíva v kontrole vstupných dát pred ich odoslaním do databázového subsystému. Pre detekciu a nahradenie nebezpečných riadiacich znakov neškodnými reťazcami je vhodné vytvoriť funkciu, ktorá v súlade s dokumentáciou k databázovému systému nájde všetky nebezpečné znaky (najmä úvodzovky a apostrofy), nahradí ich bezpečnými znakmi a na výstupe odovzdá zabezpečený reťazec. Niektoré programovacie jazyky takéto funkcie už poskytujú v rámci knižníc na prácu s vybranými databázovými systémami (napr. PHP od verzie 4). Existujú tiež programovacie jazyky, v ktorých sa o ochranu pred SQL injection stará prostredie automaticky (napr. ASP .NET).

```
function sql_safe($string)
  {
  return strtr($string,"\"\'","  ");
  }
```

```
function db_safe_string($string)
  {
  return pg_escape_string($string);
  }
```

Obr. 6.3: Pred útokmi typu SQL Injection je možné sa chrániť pomocou kontroly vstupných dát, ktoré sú ďalej interpretované databázovým systémom. Vstupné dáta je možné zabezpečiť vlastnou funkciou (vľavo) alebo použitím už existujúcej funkcie v danom prostredí (vpravo). V praxi sa odporúča používať druhé riešenie

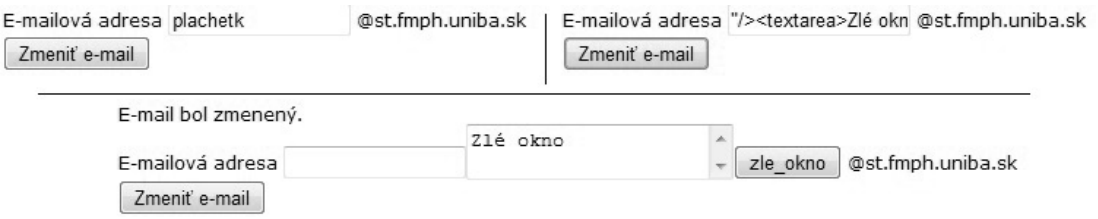
Shell Command Injection

Shell Command Injection je názov pre útok triedy code injection zameraný na shell (príkazový riadok, konzolu). „Účelom útoku Shell Command Injection je vloženie a vykonanie útočníkom špecifikovaných príkazov v zraniteľnej aplikácii. V situáciach ako je táto, aplikácia, ktorá vykoná neželané systémové príkazy, je ako pseudo shell systém, a útočník ho môže použiť ako ktorýkoľvek autorizovaný systémový užívateľ. Príkazy sú však vykonávané s rovnakými právami a v rovnakom prostredí aké má webová aplikácia. Útoky Shell Command Injection sú vo väčšine prípadov možné pre chýbajúcu kontrolu vstupných dát, s ktorými má útočník možnosť manipulovat (formuláre, cookies, HTTP hlavičky atď.).“ [OWA09]

Ochrana proti Shell Command Injection útokom spočíva na rovnakom princípe ako ochrana voči SQL injection, jediný rozdiel je v množine riadiacich znakov pre shell. Druhým spôsobom ochrany voči takýmto útokom je vyhýbanie sa volaniam externého kódu za pomoci shell-u.

6.1.2 Cross-site Scripting

Cross-site Scripting je forma útoku zameraná na výstup z aplikácie a klientský prehliadač webových stránok. Ak považujeme prehliadač webových stránok za subsystém aplikácie, potom môžeme útok zaradiť medzi útoky triedy code injection.
„Cross-site Scripting spočíva v oklamaní webového servera takým spôsobom, že potom posiela užívateľovi škodlivý kód HTML, obvykle ide o skript. Zámerom takéhoto útoku je často ukradnúť informácie o prebiehajúcej session a následne komunikovať so serverom menom obete. Skripty sa dajú tiež použiť na zmenu obsahu webových stránok, aby užívateľom zobrazili nesprávne informácie, alebo na presmerovanie formulárov tak, aby boli utajené dáta odoslané na počítač útočníka.“ [Hus04]



Obr. 6.4: Príklad nezabezpečeného formulára na zmenu e-mailovej adresy. Nová e-mailová adresa obsahuje kód, ktorý do formulára vloží nové objekty, čím ovplyvní výstup pre daného užívateľa. Ak sa e-mailová adresa zobrazuje aj v iných častiach aplikácie, útočník má možnosť upraviť obsah aj týchto častí, čím ovplyvní výstup aj pre ďalších užívateľov

Ochrana proti Cross-site Scripting útokom spočíva v kontrole výstupných dát pred ich odoslaním užívateľovi.

```
$input = filter_input(INPUT_POST, 'user_mail', FILTER_SANITIZE_SPECIAL_CHARS);
$output['user_mail'] = filter_var($output['user_mail'],FILTER_SANITIZE_SPECIAL_CHARS);
```

Obr. 6.5: Príklad funkcií na zabezpečenie formulára na zmenu e-mailovej adresy. Prvý príkaz zabezpečí premennú s e-mailovou adresou pred jej vložením do databázy. Vloženie bezpečných údajov však negarantuje, že dáta budú bezpečné aj pri ich načítaní, preto druhý riadok zabezpečí výstup z databázy

6.1.3 Packet Sniffing

Packet sniffing alebo zachytávanie packetov[2] je spôsob získavania informácií. Útočník zachytáva komunikáciu medzi klientským webovým prehliadačom a webovým serverom, v ktorej hľadá citlivé dáta ako sú prihlasovacie mená, heslá či identifikátory session. Poznáme dve formy zachytávania packetov: aktívnu a pasívnu. Pri pasívnej forme útočník komunikáciu iba odpočúva, nezasahuje do jej priebehu. Pri aktívnej forme tvorí útočník skrytého sprostredkovateľa komunikácie medzi užívateľom a serverom. Užívateľ odosiela dáta útočníkovi v presvedčení, že ide o požadovaný server, útočník dáta spracuje a dáta odošle serveru. Odpoveď zo servera sa vracia útočníkovi, ktorý môže opäť získať dôležité dáta, a ten ju z dôvodu utajenia svojej činnosti preposiela užívateľovi. Takýto útok sa nazýva Man-in-the-middle (človek uprostred). V nasledujúcom texte predpokladáme, že čitateľ je oboznámený so základnými pojmami z oblastí kryptológie a digitálneho

[2]Jednotka dát prenášaná protokolom TCP/IP.

podpisu.

Riešením ochrany informácií pred zachytávaním packetov je šifrovanie komunikácie medzi užívateľovým webovým prehliadačom a webovým serverom. „V podmienkach Internetu znamená šifrovanie väčšinou protokol HTTPS. Jednoducho povedané HTTPS je starý známy HTTP, použitý cez šifrovaný kanál. Šifrovaný kanál vytvára protokol Secure Socket Layer (SSL)[3] alebo jeho nástupcu Transport Layer Security (TLS)[4,5].“ [Hus04] Pred začiatkom požadovanej komunikácie medzi klientským prehliadačom webových stránok a webovým serverom prebieha fáza „podania rúk“ (tzv. handshake). V tejto fáze sa klientský prehliadač dohodne so serverom na šifrovacích a hašovacích algoritmoch, server predá klientovi svoj digitálny certifikát a vyberie sa symetrický šifrovací kľúč na šifrovanie komunikácie. Od tejto chvíle je komunikácia šifrovaná pomocou dohodnutého symetrického kľúča.

V prípade použitia HTTPS a adekvátnych kryptologických funkcií je komunikácia medzi užívateľom a serverom chránená pred únikom dôverných informácií a útokom man-in-the-middle. Slabým miestom takéhoto riešenia sú užívatelia, ktorí v mnohých prípadoch nekontrolujú pravosť a platnosť certifikátov. Útočník si tak môže vytvoriť vlastný neplatný certifikát. Je potrebné nielen poskytnúť zabezpečenú aplikáciu, ale aj zaškoliť jej koncových užívateľov.

6.1.4 Krádež sessions

Sessions (relácie) sa často používajú na identifikáciu prihlásených užívateľov v systéme. Ak dôjde ku skompromitovaniu identifikátora session, môže útočník prebrať spojenie iného užívateľa a vykonávať úkony v jeho mene. Identifikátor session môže útočník získať viacerými spôsobmi — uhádnutím, z hlavičky požiadavky (ako parameter metódy GET), útokom Cross-site Scripting alebo metódou Packet Sniffing.

Obranou proti uhádnutiu je generovanie náhodných identifikátorov. Identifikátor session je vhodné prenášať metódou POST, ktorej obsah sa na rozdiel od metódy GET neukladá v prehliadači. Proti Packet Sniffingu je vhodnou obranou použitie HTTPS protokolu. Súčasné webové servery podporujú tieto spôsoby ochrany.

[3] A. O. Freier, P. Karlton, P. C. Kocher: The SSL Protocol Version 3.0, 1996, http://www.mozilla.org/projects/security/pki/nss/ssl/draft302.txt

[4] T. Dierks, C. Allen: RFC 2246: The TLS Protocol Version 1.0, 1999, http://www.ietf.org/rfc/rfc2246.txt

[5] E. Rescorla: RFC 2818: HTTP Over TLS, 2000, http://www.ietf.org/rfc/rfc2818.txt

Existujú metódy, pomocou ktorých je možné významne sťažiť zneužitie session aj po prezradení identifikátora. Tieto metódy teraz v krátkosti uvedieme:

- zviazanie identifikátora session s IP adresou užívateľa
- zviazanie identifikátora s určitou časťou hlavičky HTTP (napríklad User-Agent)
- zmena identifikátora po každej požiadavke
- zneplatnenie identifikátora pri každom podozrení na jeho prezradenie

Je potrebné upozorniť, že uvedené metódy nezaručujú absolútne zabezpečenie, ale ich použitie sa vysoko odporúča, lebo kladú pred útočníka dodatočné prekážky.

```
session_regenerate_id(true);
$_SESSION['id'] = htmlspecialchars($zaznam['user_id']);
$_SESSION['type'] = htmlspecialchars($zaznam['user_type']);
$_SESSION['ip'] = htmlspecialchars($_SERVER['REMOTE_ADDR']);
...

if (isset($_SESSION['id']))
{
  //nespravna IP adresa - podozrenie na ukradnutie session id
  if ($_SESSION['ip'] != $_SERVER['REMOTE_ADDR'])
  {
    auth_log_out();
  }

  @$vysledok= db_query("SELECT user_ip_address,user_session_id
                        FROM users
                        WHERE user_id='".db_safe_string($_SESSION['id'])."'");
  @$zaznam=db_fetch_array($vysledok);

  //nespravne sessionID - pravdepodobne viacnasobna sucasna praca s 1 kontom
  if ((!$zaznam) OR (session_id() != $zaznam['user_session_id'])
      OR ($_SERVER['REMOTE_ADDR'] != $zaznam['user_ip_address']))
  {
    auth_log_out();
  }
}
```

Obr. 6.6: Ochrana identifikátora session v systéme pre podporu organizácie cvičení

Session Fixation

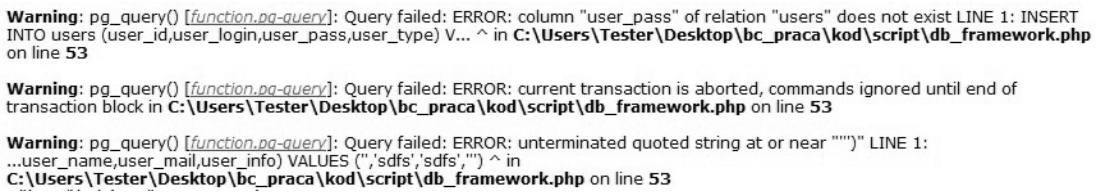
Pre získanie identity užívateľa sa útočník nemusí snažiť len o získanie už existujúceho identifikátora. Session Fixation je typ útoku, pri ktorom si útočník nechá webovým serverom vytvoriť vlastnú session a následne presvedčí užívateľa, aby pri prvej návšteve webovej stránky použil jeho identifikátor session. Po prihlásení

užívateľa tak útočník získava jeho prístupové práva k aplikácii.

Riešením ochrany pred útokmi typu Session Fixation je vygenerovanie nového identifikátora session a zneplatneniu identifikátora starého.

6.2 Hlásenia o chybách pri používaní aplikácie

Chyby, ktoré sa vyskytnú pri používaní aplikácie, môžeme rozdeliť do štyroch skupín. Prvú skupinu tvoria syntaktické chyby v kóde aplikácie, druhou skupinou sú chyby vznikajúce v dôsledku nedostupnosti služieb subsystémov, treťou skupinou sú chyby vo vstupoch od užívateľa a štvrtou sú chyby v aplikačnej logike. Priame bezpečnostné riziko tvoria chyby štvrtej skupiny, keď sa aplikácia správa inak ako to mali vývojári v úmysle. Takéto chyby môžu viesť k neoprávnenému získaniu prístupu k funkciám a dátam aplikácie a je potrebné sa im vyvarovať dôsledným návrhom a testovaním. Nepriame bezpečnostné riziko tvoria hlásenia o výskyte chýb. Webové servery zväčša podporujú tzv. odľaďovací mód, pri ktorom je užívateľ relatívne podrobne informovaný o chybe. Ak je použitý takýto mód v „ostrej“ prevádzke alebo webový server vždy podrobne informuje o chybách, dochádza k únikom informácií o štruktúre programu a jeho subsystémov (napr. databázových tabuliek), ktoré môže útočník využiť pri útoku. Chyby je preto vhodné zaznamenať do vyhradeného zabezpečeného súboru a užívateľovi odoslať iba absolútne nevyhnutné informácie o chybe.



Obr. 6.7: Príklad štandardného informovania o chybe serverom Apache. Ak sa útočníkovi podarí získať takúto informáciu o chybe, získava z nej informácie o štruktúre tabuliek a dotazov, ktoré mu môžu pomôcť v ďalších útokoch na aplikáciu. Štandardné chybové výpisy pri volaní funkcie je v PHP možné potlačiť pomocou operátora @ pred volaním tejto funkcie

Ďalším bezpečnostným rizikom je nevhodné a podrobné informovanie užívateľa o nekorektne zadanom vstupe. Ak užívateľovi poskytneme príliš veľa informácií, môže to znížiť bezpečnosť aplikácie[6].

[6]Príkladom je informovanie užívateľa o existencii prihlasovacieho mena (a prípadné ponúknutie najbližších správnych možností), ktoré znižujú počet kombinácií, ktoré musí útočník vyskúšať, aby získal prístup ku kontu užívateľa (útočník najskôr háda prihlasovacie meno a pre existujúce meno skúša heslá, nemusí však skúšať všetky kombinácie prihlasovacích mien a hesiel).

6.3 Defenzívne programovanie

„Defenzívne programovanie je založené na myšlienke, že ak časť aplikácie dostane neplatné dáta, nič sa jej nestane ani v prípade, ak sú nesprávne dáta dôsledkom chyby v inej časti aplikácie. Vo všeobecnosti sa dá povedať, že ide o snahu rozoznať, kde môže mať program problémy.“ [McC04]

Tento prístup je vhodný aj pri tvorbe webových aplikácií. Sverre H. Huseby vo svojej knihe [Hus04] uvádza niekoľko ďalších pravidiel, ktoré by mal vývojár webových aplikácií dodržiavať:

- Odosielanie požiadaviek prebieha cez metódu POST, ak majú vedľajšie účinky (úprava databázy, zmena súborov,...). Webové prehliadače pri metóde POST žiadajú potvrdenie opätovného odoslania, pri metóde GET môžu byť požiadavky odoslané aj bez vedomia užívateľa (funkcie Dopredu a Späť, obnova stránky a pod.)

- Server nepozná bezpečného klienta. Všetky požiadavky vrátane výsledkov skriptovacích jazykov môžu byť na strane klienta upravené, preto je potrebné všetky dáta overovať s údajmi na serveri.

- Dokumentácia subsystémov tretích strán nemusí byť správna. Správanie takýchto subsystémov je vhodné testovať proti dokumentácii.

- Pri filtrovaní neplatných dát je lepšie použiť whitelist než blacklist. Whitelist je zoznam všetkých povolených a platných dát, ostatné dáta sú považované za neplatné. Blacklist je zoznam všetkých neplatných dát, všetky ostatné dáta sú považované za platné.

- Úprava neplatných vstupných dát na platné dáta je nebezpečná. Ak útočník odhalí algoritmus tejto úpravy, môže ho využiť pri svojom útoku.

- Kontrola prístupových práv či identity užívateľa nesmie prebiehať formou skriptov na strane klienta. Vykonávanie skriptov môže klient zakázať alebo pozmeniť podľa svojich potrieb. Kontrola platnosti vstupných dát z rovnakých dôvodov nesmie prebiehať len na strane klienta[7].

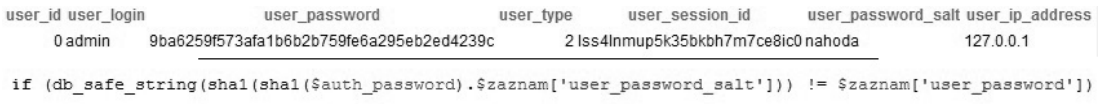
- Pre vstupné dáta generované serverom (ako sú napr. zoznamy položiek alebo skryté polia formulárov) je vhodné použiť nepriamu adresáciu dát pomocou číselníkov. Ak sú dátami časti URL adries (napríklad názvy súborov), útočník nebude mocť tieto dáta získať zo zdrojového kódu webstránky.

[7] Z dôvodu zvýšenia užívateľského komfortu a odozvy systému je vhodné ju však použiť ako na strane servera, tak aj na strane klienta.

- Heslá nikdy nie sú ukladané v nešifrovanej podobe. Útočník aj v prípade získania údajov z databázy nevie odhaliť heslá užívateľov, ak je použitá „bezpečná“[8] šifrovacia funkcia. Častým spôsobom pre bezpečné uchovávanie hesiel je pridanie náhodne vygenerovaného znakového reťazca (tzv. soľ) k heslu užívateľa a aplikovanie hashovacieho algoritmu na takýto reťazec. Výsledný reťazec je potom spolu so „soľou“ uložený do databázy. Medzi dobré vlastnosti takéhoto riešenia patrí aj nemožnosť identifikácie užívateľov s rovnakými heslami.

- Metóda GET neslúži na odosielanie dôležitých údajov (napr. identifikátory session, chránené informácie, atď.). Takéto údaje sa ukladajú na rôznych miestach a slúžia napríklad na vytváranie histórie navštívených stránok.

- Utajenie zdrojového kódu aplikácie nie je ochranou pred útokmi. Vývojár musí počítať s tým, že sa útočník dostane k zdrojovým kódom aplikácie, a prispôsobiť tomu návrh aplikácie.



Obr. 6.8: V systéme pre podporu organizácie cvičení nie sú heslá uklada v pôvodnej textovej podobe. Atribút user_password_salt obsahuje náhodne generovaný reťazec (soľ), ktorý sa pridá k digitálnemu odtlačku hesla (výsledok hashovacej funkcie) a do atribútu user_password sa uloží digitálny odtlačok výsledného reťazca. Na obrázku sa nachádza výstup z databázy a príkaz overujúci platnosť takto uloženého hesla

Našim cieľom bolo oboznámiť sa v súčasnosti s najznámejšími a najčastejšie zneužívanými chybami v zabezpečení webových aplikácií a ochranou pred nimi. Problematika bezpečnosti webových aplikácií je značne rozsiahla oblasť vedy, ktorej sa venujú mnohé organizácie - OWASP[9], SANS[10], W3C[11], WASC[12] a ďalšie.

[8] Aktuálne považovaná za bezpečnú, prelomiteľnú len úplným preberaním všetkých možností.

[9] The Open Web Application Security Project, http://www.owasp.org

[10] The SANS (SysAdmin, Audit, Network, Security) Institute, http://www.sans.org

[11] The World Wide Web Consortium, http://www.w3.org/Security/

[12] Web Application Security Consortium, http://www.webappsec.org/

Kapitola 7

Databázové systémy

V súčasnosti sa máloktorá webová aplikácia zaobíde bez použitia databáz. V tejto kapitole stručne definujeme základné pojmy z oblasti databázových systémov a uvedieme niektoré nástroje za účelom efektívneho návrhu databázy pre webové aplikácie.

„**Databáza** je kolekcia dát udržiavaná a prístupná pomocou počítača.“ [Kri92]

„**Databázový systém** je nástroj pre efektívne vytváranie a spravovanie veľkých objemov perzistentných dát. Od databázového systému sa očakáva, že:

- Umožní užívateľom vytvárať nové databázy a špecifikovať ich schémy (logické štruktúry dát) pomocou špeciálneho jazyka na definíciu dát.
- Dá užívateľom možnosť dotazovať sa na dáta a upravovať ich pomocou dotazovacieho jazyka.
- Podporuje uchovávanie veľmi veľkých objemov dát po dlhú dobu umožňujúc efektívny prístup k dátam (pomocou dotazov) a úpravy databáz.
- Je odolná voči zlyhaniam, chybám a nesprávnemu používaniu.
- Riadi súčasný prístup k dátam od mnohých užívateľov, pričom garantuje izoláciu[1] a atomicitu dotazov[2].“ [UW01]

[1] Užívateľské dotazy nemajú neočakávaný vplyv na iných užívateľov.
[2] Akcia vo forme dotazu sa buď vykoná celá alebo vôbec.

7.1 Transakcie

Pomocou dotazovacieho jazyka máme možnosť vykonávať atomické operácie nad databázou. Vieme teda vytvoriť, upraviť alebo odstrániť tabuľku, pridať, upraviť alebo odstrániť jej obsah. Prax ukázala, že okrem samostatných atomických dotazov je v mnohých prípadoch požadované atomické vykonanie určitej skupiny operácií[3]. Z tohto dôvodu bola do databázových systémov zapracovaná podpora transakcií (transakčný systém).

Transakcia je skupina operácií, ktorá musí byť vykonaná atomicky a izolovane od ostatných transakcií.

Požiadavky na transakčný systém je možné vyjadriť skratkou ACID:

- **Atomicita (Atomicity).** Transakcia je vykonaná ako celok alebo nie je vykonaná vôbec.
- **Konzistencia (Consistency).** Transakcia zachováva konzistenciu dát v databáze. Vykonanie transakcie teda nenaruší podmienky a obmedzenia kladené na jednotlivé časti databázy.
- **Izolovanosť (Isolation).** Transakcia nie je ovplyvňovaná a neovplyvňuje priebeh a výsledok vykonávania iných transakcií vykonávaných v rovnakom čase.
- **Trvanlivosť (Durability).** Výsledok úspešne vykonanej transakcie sa nesmie stratiť, musí byť uchovaný.

Vývojár webovej aplikácie sa stará len o splnenie požiadavky konzistencie. O splnenie ostatných požiadaviek sa stará transakčný systém.

Úlohy procesora transakcií v databázovom systéme:

- **Uchovávanie záznamov.** Uchovávanie informácií o zmenách v databáze s cieľom zaručiť trvanlivosť zmien v databáze a odolnosť voči výpadkom databázového systému.
- **Plánovanie transakcií.** Naplánovanie jednotlivých operácií z transakcií pri súčasnom spracovaní viacerých transakcií s dôrazom na izolovanosť transakcií.
- **Riešenie uviaznutí.** Vyhýbanie sa alebo riešenie problému uviaznutia transakcií, ktorý môže vzniknúť pri súčasnom spracovávaní viacerých transakcií požadujúcich rovnaké zdroje dát.

[3]Napríklad pre elektronické prevody financií medzi 2 účtami je kriticky dôležité, aby skupina príkazov na zníženie zostatku u odosielateľa a zvýšenie u prijímateľa bola vykonaná atomicky.

7.2 Relačné databázy

V prvých databázových systémoch mali dáta z pohľadu používateľa rovnakú štruktúru ako ich fyzická reprezentácia v systéme. Existovalo viacero dátových modelov takýchto databáz (hierarchický model, sieťový model), avšak ich hlavnou nevýhodou bolo nepodporovanie vyšších programovacích jazykov. Objavila sa tak potreba oddelenia užívateľského pohľadu na štruktúru dát od ich fyzickej reprezentácie pomocou koncepčných dátových modelov, ktoré budú použiteľné aj vo vyšších programovacích jazykoch. Poznáme viacero koncepčných dátových modelov — entitno-relačný, relačný, navigačný (XML), objektový. V súčasnosti najpoužívanejším koncepčným dátovým modelom je relačný dátový model, s ktorým v roku 1970 prišiel Edgar F. Codd [Cod70].

V relačnom databázovom modeli sú dáta logicky usporiadané do tabuliek nazývaných relácie. Ako dotazovací jazyk nad relačnými databázami sa presadil jazyk SQL[4] (Structured Query Language). Medzi najpoužívanejšie databázové systémy s podporou jazyka SQL patria Microsoft SQL Server, MySQL, Oracle, PostgreSQL, Sybase a ďalšie.

7.2.1 Normalizácia databázy

Normalizácia databázy je proces transformácie pôvodného návrhu databázy do normalizovaného tvaru s cieľom minimalizovať redundantné dáta. Redundantné dáta zvyšujú požiadavky na systémové zdroje pre ich uchovávanie, komplikujú aktualizáciu dát a zvšujú riziko nekonzistencie omylom vývojára. V tejto časti zadefinujeme niektoré pojmy a uvedieme základné normálové formy.

Atribút relácie je usporiadaná dvojica [názov, doména/typ prvku]. [EN03]

Relácia je podmnožina kartézskeho súčinu množín, kde množiny sú domény atribútov relácie. Pre ďalšiu potrebu budeme množinu atribútov považovať za usporiadanú podľa názvov atribútov. Reláciu môžeme graficky reprezentovať dvojrozmernou tabuľkou, v ktorej stĺpce sú atribúty relácie a riadky tvoria prvky relácie. [EN03]

Nadkľúč relácie je podmnožina atribútov relácie, ktorá jednoznačne určuje všetky atribúty relácie. [EN03]

Kľúč relácie je nadkľúč relácie, ktorý prestáva byť odobratím ľubovoľného atribútu z nadkľúča nadkľúčom relácie. Kľúč relácie je teda minimálny kľúč a 2 rôzne prvky relácie nemajú rovnaké ohodnotenie pre všetky atribúty kľúča.

[4] ISO/IEC 9075:2008 (SQL:2008)

[EN03]

Cudzí kľúč relácie je podmnožina atribútov relácie, ktorá referencuje druhú reláciu, ak atribúty v cudzom kľúči sú z rovnakých domén ako atribúty kľúča v druhej relácii a ohodnotenie cudzieho kľúča z prvej relácie je buď zhodné s ohodnotením pre niektorý prvok druhej relácie alebo je nedefinované. [EN03]

Primárny kľúč je práve jeden kľúč relácie, ak existuje viacero kľúčov, jeden je vybraný za primárny a ostatné sa považujú za sekundárne kľúče. V praxi sa ako primárny kľúč volí nový jednoznačný atribút (zväčša reprezentovaný číslom, ktoré sa zvyšuje). [EN03]

Funkčná závislosť $\mathcal{X} \to \mathcal{Y}$, kde $\mathcal{X}$, $\mathcal{Y}$ sú množiny atribútov, platí v relácii práve vtedy, keď pre 2 ľubovoľné prvky z relácie platí, že ak sa zhodujú ich hodnoty na množine $\mathcal{X}$, tak sa zhodujú aj na $\mathcal{Y}$. Funkčná závislosť je úplná, ak odstránením ľubovoľného atribútu z $\mathcal{X}$ funkčná závislosť zaniká. [EN03]

Relačná schéma je usporiadaná dvojica [relácia, množina funkčných závislostí platiacich v relácii]. [EN03]

Prvá normálová forma

Relačná schéma je v prvej normálovej forme práve vtedy, keď žiaden atribút relačnej schémy nie je zložený atribút.

Druhá normálová forma

Relačná schéma je v druhej normálovej forme práve vtedy, keď každý atribút je súčasťou primárneho kľúča relácie alebo je úplne funkčne závislý od primárneho kľúča.

Tretia normálová forma

Relačná schéma je v tretej normálovej forme, ak je v druhej normálovej forme a pre každú platnú funkčnú závislosť $\mathcal{X} \to \mathcal{Y}$ v relácii je $\mathcal{X}$ nadkľúč relácie alebo $\mathcal{Y}$ je časťou niektorého kľúča relácie.

Boyce-Coddova normálová forma

Relačná schéma je v tretej normálovej forme, ak je v druhej normálovej forme a pre každú platnú funkčnú závislosť $\mathcal{X} \to \mathcal{Y}$ v relácii je $\mathcal{X}$ nadkľúč relácie.

7.3 Databázy a webové aplikácie

Výber správneho databázového systému, dotazovacieho jazyka a návrh relačných schém sú dôležitou súčasťou návrhu architektúry webovej aplikácie, pre potrebu zadefinovania niektorých pojmov sme ho však zaradili až do tejto kapitoly.

Prvým krokom je teda výber vhodného databázového systému, nad ktorého databázou bude webová aplikácia pracovať. Vzhľadom na masívne rozšírenie relačných databázových systémov je nepravdepodobné, že pri tvorbe webovej aplikácie by v súčasnosti bol vybraný iný ako relačný databázový model a dotazovací jazyk SQL. Mnohé databázové systémy ponúkajú rozšírenia jazyka SQL (napríklad procedurálny jazyk PL/SQL od Oracle), ich kompatibilita s inými databázovými systémami je otázna. Pri rozhodovaní sa o výbere databázového systém však treba brať do úvahy aj tieto rozšírenia. Ak navrhujeme databázu využívajúcu len základné možnosti jazyka SQL, môžeme použiť ľubovoľný databázový systém, z finančného hľadiska sa ako dobrá voľba javia databázy MySQL, PostgreSQL a Microsoft SQL Server Express.

Pre systém pre podporu organizácie cvičení bol primárne vybraný databázový systém PostgreSQL, avšak vďaka používaniu iba základných dotazov a návrhu subsystému pre komunikáciu s databázovým systémom je možné aplikáciu v krátkom čase nasadiť s ľubovoľným databázovým systémom.

7.3.1 Návrh databázy

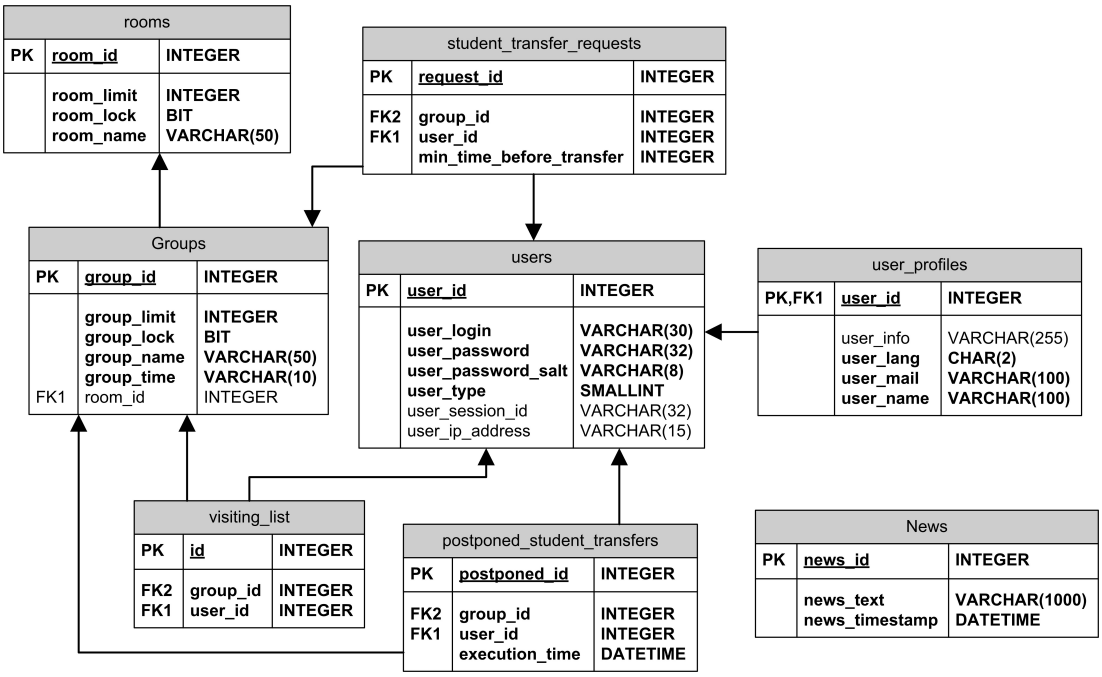
Nasledujúce kritéria považujeme za dôležité pri návrhu databázy (ale aj jej výbere). [EN03]

- **Odozva** je čas potrebný na spracovanie a odoslanie výsledku transakcie. Závisí od databázového systému a jeho prístupovej dobe k uloženým dátam, aktuálneho zaťaženia systému a servera, na ktorom je databázový systém nasadený.
- **Objemnosť** označuje množstvo úložného priestoru potrebného na uloženie databázy v systéme. Pri optimalizácii úložného priestoru je dôležité odstrániť redundantné dáta z databázy, pokiaľ ich odstránením sa výrazne nezvýši doba odozvy. Pri odstraňovaní redundantných dát je užitočná dekompozícia relačných schém do normálových foriem. Konkrétne algoritmy pre dekompozíciu do normálových foriem ako i ďalšie normálové formy sa nachádzajú v knihe [EN03].
- **Priepustnosť transakcií** je priemerný počet transakcií za minútu, ktoré databázový systém dokáže spracovať.

Návrh databázy by sa mal:

- Vyhýbať redundancii dát. Redundancia zvyšuje požiadavky na úložný priestor, pri modifikácii hodnôt v databáze zvyšuje riziko nekonzistencie dát a počet operácií, ktoré je potrebné vykonať.
- Vyhýbať slabým reláciam. Relácia je slabá, ak neobsahuje atribút, ktorý je jej kľúčom, ale existuje iná relácia, ktorej atribút je cudzím kľúčom slabej relácie. Výskyt slabých relácií v databáze ma za následok vznik anomálií a stratu informácií.
- Minimalizovať počet relácií v databáze. Ak je relácia nahraditeľná vytvorením atribútu v inej relácii a všetky relačné schémy zostanú v normálovej forme, je vhodné takúto zmenu vykonať.

Pre vizualizáciu návrhu riešenia koncepčného databázového modelu sa používajú entitno-relačné diagramy a UML class diagramy. Bližšie informácie a spôsob modelovania diagramov sa nachádzajú v knihe [EN03].



Obr. 7.1: Entitno-relačný diagram návrhu koncepčného databázového modelu systému pre podporu organizácie cvičení

7.3.2 Dotazy nad databázou

Databázy potrebujeme nielen vytvárať, ale ich aj upravovať a čerpať z nich informácie. Tvorba optimalizovaných dotazov je podrobne popísaná v knihe [EN03],

uvádzame preto na záver iba niekoľko poznatkov, ku ktorým sme prišli pri stavbe systému pre podporu organizácie cvičení:

- Pred použitím databázových príkazov je potrebné si naštudovať dokumentáciu ku konkrétnemu databázovému systému. Príkazy sa totiž v mnohých prípadoch môžu syntakticky odlišovať. Príkaz CREATE TABLE, ktorý vytvára novú tabuľku má zvyčajne pre každý databázový systém inú syntax v závislosti na podporovaných dátových typoch, ich názvoch a ďalších faktoroch.

- Pri komplikovaných dotazoch je v mnohých prípadoch vhodné použiť dočasné tabuľky a pohľady. Správnym použitím je možné sa vyvarovať konštrukcii neprehľadných dotazov. Nevýhodou dočasných tabuliek a pohľadov je nekompatibilná syntax medzi rôznymi databázovými systémami.

- Mnohé databázové systémy implementujú niektoré často používané funkcie, ktoré je možné volať vrámci SQL dotazu. Niektoré z týchto funkcií sú definované aj v rámci štandardu ISO SQL:2008. Nevýhodou je opäť možnosť nekompatibilnej syntaxe medzi rôznymi databázovými systémami, prípadne chýbajúca implementácia v iných databázových systémoch. Medzi často používané funkcie patria napríklad NOW(), LOWER(string), UPPER(string) a ďalšie.

- Databázové systémy môžu podporovať viacero formátov ukladania dát, ktoré sa môžu líšiť v rýchlosti spracovania dotazov ale aj v podpore niektorých vlastností. Medzi najpoužívanejšie formáty v databázovom systéme MySQL patria MyISAM a InnoDB. V súčasnosti medzi hlavné rozdiely medzi MyISAM a InnoDB patria podpora transakcií a podpora cudzích kľúčov, ktoré v MyISAM nie sú zahrnuté. Pri výbere databázového systému je teda potrebné vybrať aj vhodný formát pre ukladanie dát podľa požadovaných vlastností.

- Dátové typy atribútov je možné ohraničiť určitým rozsahom, prípadne existuje niekoľko verzií dátových typov. Ak sú známe ohraničenia hodnôt atribútov, je dobré vybrať vhodný dátový typ, čím dôjde k úspore úložného miesta.

- Textový reťazec pevnej dĺžky je vhodné použiť ako dátový typ len v prípade, že slúži na uloženie dát, u ktorých máme garantovaný požadovaný tvar. V ostatných prípadoch je vhodnejšie použitie textového reťazca s dynamickou dĺžkou zhora ohraničenou rozumnou hodnotou[5].

[5]Príkladom nesprávneho použitia textových reťazcov pevnej dĺžky môže byť atribút reprezentujúci identifikátor relácie prihláseného užívateľa (viď kapitola Webové aplikácie a bezpečnosť). Počas stavby aplikácie bolo zistené, že PHP funkcia session_id() vracia textový reťazec s dĺžkou 32 znakov, preto bolo rozhodnuté o použití dátového typu textového reťazca

```
function user_delete($id) // returns bool: 0 - OK | -1 - user does not exist
  {
  $id = db_safe_string($id);
  db_begin();
  $result = db_query("SELECT user_id FROM users WHERE user_id = $id");
  if (db_num_rows($result) > 0)
    {
    db_query("DELETE FROM visiting_list WHERE user_id = $id");
    db_query("DELETE FROM postponed_transfers WHERE user_id = $id");
    db_query("DELETE FROM transfer_requests WHERE user_id = $id");
    db_query("DELETE FROM user_profiles WHERE user_id = $id");
    db_query("DELETE FROM users WHERE user_id = $id");
    db_commit();
    return 0;
    }
  db_abort();
  return -1;
  }
```

Obr. 7.2: Príklad využitia transakcií — korektné odstránenie užívateľa zo systému. Transakčný systém zabezpečí konzistenciu dát v databáze aj v prípade výpadku databázového systému počas vykonávania transakcie

```
$query = "CREATE TEMPORARY TABLE almost_visiting_student_list ON COMMIT DROP AS
            (SELECT user_id
             FROM visiting_list
             WHERE group_id='".$_POST['group_edit']."'
               UNION
             SELECT user_id
             FROM postponed_transfers
             WHERE group_id='".$_POST['group_edit']."'
            );";
$query .= "SELECT user_id,user_name
          FROM user_profiles
          WHERE user_id > 0
           AND user_id NOT IN (SELECT user_id FROM almost_visiting_student_list)
          ORDER BY user_name ASC;";
$res2 = db_query($query);
```

Obr. 7.3: Dočasná tabuľka sprehľadňuje komplexnejší dotaz na databázu. Voľba ON COMMIT DROP zabezpečí odstránenie dočasnej tabuľky z databázového systému po skončení transakcie

pevnej dĺžky. Aplikácia pri nasadení fungovala správne, došlo však k výmene webového servera a zrazu sa nebolo možné prihlásiť. Dôvodom bolo iné nastavenie webového servera, ktoré malo za následok, že funkcia session_id() vracala textové reťazce už iba dĺžky 27 znakov, do databázy boli reťazce doplnené na 32 znakov, overenie zhody identifikátora relácie vždy skončilo neúspechom a užívateľ bol okamžite po prihlásení z bezpečnostných dôvodov odhlásený.

Kapitola 8

Stavba aplikácie

V tejto kapitole sa venujeme fáze stavby aplikácie. Neuvádzame konkrétne postupy pre konkrétne programovacie jazyky, našim cieľom nie je ani vysvetlenie niektorého z programovacích jazykov, namiesto toho uvádzame niekoľko nástrojov, ktoré napomáhajú pri stavbe aplikácie a sú univerzálne použiteľné, spolu s príkladmi ich využitia.

8.1 Prototypy

Rovnako ako pri návrhu aplikácie, tak aj pri jej stavbe je v mnohých prípadoch vhodné použiť prototypy, ktorým sme sa venovali v kapitole 4.3.1 Prototypy.

```
function auth_issignin()
{
  return true;
}
```

```
function auth_userid()
{
  return 12345;
}
```

```
function auth_usertype()
{
  return -1;
}
```

Obr. 8.1: Tri funkcie z autentifikačného modulu vo forme prototypov, ktoré ihned po zavolaní vrátia predvolenú hodnotu

8.2 Trasovací kód

V prípadoch, keď si nemôžeme dovoliť zanedbať niektoré detaily, môže byť vhodnejšie použiť namiesto prototypov trasovací kód. Pod pojmom „trasovací kód“ rozumieme inkrementálny prístup[1] k vývoju niektorej časti aplikácie. Trasovací kód na rozdiel od prototypov nie je dočasným kódom, ktorý je neskôr zahodený, ale tvorí základ pre ďalší vývoj. Obsahuje všetky kontroly chýb, náležitú štruktúru, dokumentáciu a rovnakú sebakontrolu, ako má finálny kód. Na rozdiel od

[1] Viď kapitola 2.

finálneho kódu nemajú implementovanú kompletnú funkcionalitu. Po napojení trasovacieho kódu je možné si overiť či sa kód blíži k požadovanému výsledku a vykonať prípadné úpravy návrhu. [HT99] Hlavnými výhodami trasovacieho kódu je postupná tvorba štruktúry, na ktorej je možné ďalej stavať, možnosť vidieť čiastkové výsledky už počas stavby a na základe týchto čiastkových výsledkov možnosť reagovať na prípadné zmeny.

```
function handle_requests()
{
  db_begin();

  $result = db_query("SELECT user_id, post_id, group_id
                      FROM student_transfer_requests
                      ORDER BY post_id ASC");
  while ($group = db_fetch_array($result))
  {
    $user_id = $group['user_id'];
    $group_id = $group['group_id'];

    if (group_free_places($group_id) > 0)
    {
      move_user_to_group($user_id, $group_id);
      db_query("DELETE FROM transfer_requests
                WHERE post_id=".$group['request_id']);
    }
  }
  db_commit();
}
```

```
function handle_requests()
{
  db_begin();

  $result = db_query("SELECT user_id, post_id, group_id
                      FROM student_transfer_requests
                      ORDER BY post_id ASC");
  while ($group = db_fetch_array($result))
  {
    $user_id = $group['user_id'];
    $group_id = $group['group_id'];

    if (group_free_places($group_id) > 0)
    {
      send_transfer_notification_email($user_id, $group_id);
      move_user_to_group($user_id, $group_id);
      db_query("DELETE FROM transfer_requests
                WHERE post_id=".$group['request_id']);
    }
  }
  db_commit();
}
```

Obr. 8.2: Príklad použitia trasovacieho kódu pri tvorbe funkcie spravujúcej žiadosti študentov na zaradenie do vybranej skupiny. Prvá generácia funkcie (vľavo) zaradí študenta do vybranej skupiny, ak je v nej voľné miesto. V druhej generácii funkcie (vpravo) bola implementovaná e-mailová notifikácia študenta o jeho zaradení do skupiny

```
function handle_requests()
  {
  db_begin();
  $result = db_query("SELECT user_id, post_id, group_id
                      FROM student_transfer_requests
                      ORDER BY post_id ASC");
  while ($group = db_fetch_array($result))
    {
    $user_id = $group['user_id'];
    $group_id = $group['group_id'];
    if (group_free_places($group_id) > 0)
      {
      send_transfer_notification_email($user_id, $group_id);
      move_user_to_group($user_id, $group_id);
      db_query("DELETE FROM student_transfer_requests
                WHERE request_id=".$group['request_id']);
      }
    else
      {
      /* Find someone from the requested group who wants to swap groups. */
      $cq = db_query("SELECT V1.group_id, V2.user_id, T2.request_id
                      FROM visiting_list V1, visiting_list V2, student_transfer_requests T2
                      WHERE V1.user_id = $user_id AND
                      V2.group_id = $group_id AND V2.user_id = T2.user_id
                      ORDER BY post_id ASC");
      if ($c = db_fetch_array($cq))
        {
        send_transfer_notification_email($user_id, $group_id);
        send_transfer_notification_email($c['user_id'], $c['group_id']);
        move_user_to_group($user_id, $group_id);
        move_user_to_group($c['user_id'], $c['group_id']);
        db_query("DELETE FROM student_transfer_requests
                  WHERE request_id=".$group['request_id']." OR request_id=".$c['request id']);
        }
      }
    }
  db_commit();
  }
```

Obr. 8.3: Tretia generácia funkcie obsahuje funkcionalitu predchádzajúcich generácií a navyše umožňuje vzájomnú výmenu 2 študentov aj v prípade, ak je jedna zo skupín plne obsadená

8.3 Jazyková lokalizácia

V prípade, že vo webovej aplikácii sa vyžaduje podpora viacerých jazykov alebo by sa v budúcnosti mohla takáto požiadavka objaviť, je dobré oddeliť statické texty od kódu aplikácie a zoskupiť ich do vyhradených súborov vo forme konštánt alebo premenných. Na pridanie podpory nového jazyka do webovej aplikácie následne stačí preklad takýchto lokalizačných súborov a import správnej lokalizácie.

```
<form action="" enctype="multipart/form-data" method="post" id="password_change_form">
  <fieldset>
    <legend><? echo $password_change_lang['1'];?></legend>
```

```
$password_change_lang = array ( "1" => "Zmena hesla",
                "2" => "Pôvodné heslo",
                "3" => "Nové heslo",
                "4" => "Nové heslo (potvrdenie)",
```

Obr. 8.4: Ukážka uchovávania textových reťazcov v externom súbore. V prípade potreby použitia iného jazyka stačí preložiť tieto texty a naimportovať správny súbor (použitie návrhového vzoru Most, resp. Stratégia)

8.4 Refaktoring

„Refaktorovanie (reštrukturalizácia) kódu je zmena vo vnútornej štruktúre softvéru, ktorá má uľahčiť pochopenie kódu a zjednodušiť (a zlacniť) následné opravy bez nutnosti zmeniť uhol pohľadu na kód.“ [McC04] Dôvodmi pre refaktorovanie môžu byť napríklad:

- Duplicitný kód je kandidátom na vytvorenie novej funkcie.
- Dlhé metódy, ktoré je zvyčajne možné rozdeliť na niekoľko kratších metód, ktoré zvyšujú modularitu aplikácie.
- Telo veľmi dlhého cyklu je pre sprehľadnenie dobrým kandidátom na novú metódu.
- Trieda využíva vnútornú implementáciu inej triedy, trieda má verejné dátové členy — riešením je lepšie zapúzdrenie a vytvorenie rozhraní.
- Nezrozumiteľný, nesprávny alebo nevhodný názov triedy, metódy alebo premennej.
- Zbytočné používanie globálnych premenných.
- Statické reťazce alebo čísla (napr. 3.14) je vhodné nahradiť symbolickými konštantami (PI = 3.14).

Refaktorovanie kódu sa odporúča robiť postupne po jednodlivých krokoch, úprava viacerých bodov súčasne môže viesť k chybám. Mnohé vývojové prostredia ponúkajú vstavané nástroje pre refaktorovanie kódu.

```
function handle_requests()   {    db_begin();
  $re = db_query("SELECT user_id, post_id, group_id, class_id
  FROM transfer_requests
   ORDER BY post_id ASC");
  while ($vy = db_fetch_array($re))    {
    $u_id = $vy['user_id'];
    $g_id = $vy['group_id'];
    $c_id = $vy['class_id'];
    if (group_free_places($g_id) > 0)     {
      $gnameq = db_query("SELECT class_name, group_name, group_time
      FROM groups NATURAL JOIN classes
      WHERE group_id='$g_id'");
      $gname = db_fetch_array($gnameq);
      $emailaddrq = db_query("SELECT user_name, user_mail, user_info
      FROM user_profiles
      WHERE user_id = '$u_id'");
      $emailaddr = db_fetch_array($emailaddrq);
      mail("plachetk@dcs.fmph.uniba.sk",
      "UDDB (".$gname['class_name']."): Presun do skupiny ".$gname['group_name'],
      "Uvod do DB systemov (".$gname['class_name']."): Vas presun do skupiny ".
      $gname['group_name']." (".$gname['group_time'].") bol pravdepodobne
      uspesny. Svoj aktualny rozvrh najdete v rezervacnom systeme
      http://cvika.dcs.fmph.uniba.sk/~plachetk/SYSCVIKA/index.php\n".$u_id."
      |".$emailaddr['user_name']."|".trim($emailaddr['user_mail'])."\n");
      if (trim($emailaddr['user_mail']) != "") {
        mail(trim($emailaddr['user_mail']."st.fmph.uniba.sk"),
        "UDDB (".$gname['class_name']."): Presun do skupiny ".$gname['group_name'],
        "Uvod do DB systemov (".$gname['class_name']."): Vas presun do skupiny
        ".$gname['group_name']." (".$gname['group_time'].") bol pravdepodobne
        uspesny. Svoj aktualny rozvrh najdete v rezervacnom systeme
        http://cvika.dcs.fmph.uniba.sk/~plachetk/SYSCVIKA/index.php\n");
      }
      move_user_to_group($u_id, $g_id);
      db_query("DELETE FROM transfer_requests WHERE post_id=".$vy['post_id']);
    }            }               db_commit();     }
```

```
function handle_requests()
  {
  db_begin();

  $result_query = db_query("SELECT user_id, post_id, group_id, class_id
                            FROM student_transfer_requests
                            ORDER BY post_id ASC");

  while ($row = db_fetch_array($result_query))
    {
    $user_id = $row['user_id'];
    $group_id = $row['group_id'];
    $class_id = $row['class_id'];

    if (group_free_places($group_id) > 0)
      {
      send_transfer_notification_email($user_id, $group_id);
      move_user_to_group($user_id, $group_id);
      db_query("DELETE FROM student_transfer_requests
                WHERE request_id=".$row['request_id']);
      }

    }

  db_commit();
  }
```

Obr. 8.5: Názorná ukážka reštrukturalizácie neprehľadného zdrojového kódu (vľavo) na relatívne malý a zrozumiteľný kód (vpravo)

8.5 Správa verzií

S pribúdajúcimi verziami aplikácie a najmä pri viacnásobných úpravách niektorých jej častí sa cenným nástrojom stáva nástroj pre správu verzií. Správa verzií spočíva v evidovaní zmien v aplikácii, pravidelnom zálohovaní a archivovaní aktuálneho kódu aplikácie pre potreby obnovy kódu do predchádzajúceho stavu. Na trhu existujú viaceré riešenia pre správu verzií, avšak pre bežné potreby je možné použiť aj jednoduchú kombináciu plánovača úloh operačného systému s komprimačným nástrojom.

```
tester@stargate:/tmp$ crontab -l
# m h  dom mon dow   command
0 12 * * * zip -r /tmp/directory`/bin/date +\%d\%m`.zip /tmp/directory
```

Obr. 8.6: Výstup z konfiguračného súboru z plánovača úloh Cron. Každý deň o 12:00 je naplánované spustenie komprimácie priečinka /tmp/directory do zip archívu, ktorého názov je zložený z názvu priečinka a dátumu, kedy bol vytvorený

8.6 Úprava zdrojového kódu

Úprava zdrojového kódu nemá dosah na výslednú funkcionalitu webovej aplikácie, jej cieľom je upraviť zdrojový kód do takej podoby, aby bol zrozumiteľný nielen pre počítače, ale aj pre ďalších vývojárov. Dobré rozvrhnutie zdrojového kódu je subjektívnym estetickým pocitom vývojára, avšak malo by sa snažiť dodržiavať štyri kritéria [McC04]: vyjadrenie logickej štruktúry kódu, konzistentná reprezentácia stavby kódu, lepšia čitateľnosť a odolnosť voči úpravám kódu.

Na úpravu zdrojového kódu sa používajú prázdne miesta (medzery, odsadenie, prázdne riadky) a zátvorky (pre zrozumiteľné označenie poradia vyhodnocovania).

8.6.1 Konvencie

Konvencie úpravy zdrojového kódu môžu byť závislé od použitého programovacieho jazyka alebo od zvyklostí vývojárskeho tímu. Uvádzame však niekoľko bodov, ktoré by mali zdrojové kódy spĺňať:

- **Čisté bloky.** Programovacie jazyky zväčša obsahujú blokové štruktúry, ktoré sú ohraničené svojim začiatkom a koncom. Je dobré takéto štruktúry oddeliť od zvyšku kódu prázdnymi riadkami a ich obsah odsadiť.
- **Logické bloky.** Logický blok (skupina súvisiacich príkazov) by mal byť oddelený prázdnymi riadkami, podobne ako odseky v knihách.

- **Jednopríkazové bloky.** Bloky kódu obsahujúce jediný príkaz by mali byť v celej aplikácii formátované rovnakým spôsobom.
- **Zložené podmienky.** Zložené podmienky je dobré rozdeliť do riadkov podľa porovnávaného obsahu.
- **Dĺžka príkazov.** Na jednom riadku by mal byť maximálne jeden príkaz o dĺžke približne 80 znakov. Dlhšie príkazy je vhodné kvôli čitateľnosti rozdeliť do viacerých riadkov.
- **Komentáre.** Komentáre by mali byť odsadené na úrovni komentovaného kódu a oddelené od kódu prázdnymi riadkami.

```
if (auth_issignin())
  {
  if ($db_handler = db_connect($db_host,$db_name,$db_user,$db_password,$db_schema))
    {
    switch($_GET['act'])
      {
      case "87":

        $user = auth_userid();

        if (isset($_POST['user_change_mail']))
          {
          $user_mail = $_POST['user_mail'];
          if (strpos($user_mail, '@'))
            {
            echo "Nespravna email adresa (uvedte svoj fakultny loginname, ".
                 "bez '@st.fmph.uniba.sk')";
            }
          else
            {
            db_begin();
            db_query("UPDATE user_profiles
                      SET user_mail='$user_mail'
                      WHERE user_id='$user'");
            echo "E-mail bol zmenený.";
            db_commit();
            }
        }

        $query = db_query("SELECT user_mail
                           FROM user_profiles
                           WHERE user_id='$user'");
        $row = db_fetch_array($query);
```

Obr. 8.7: Príklad úpravy zdrojového kódu dodržiavajúcej konvencie. Z kódu je zrejmé, že užívateľom poskytuje možnosť zmeny svojej e-mailovej adresy

Kapitola 9

Finalizačné procesy

V poslednej kapitole sa venujeme spôsobom testovania, ladenia kódu aplikácie a dokumentácie systému.

9.1 Testovanie a ladenie

„Testovanie je nástroj pre detekciu chýb.“ [McC04] Z pohľadu vývojára sú dôležité tri druhy testovania:

- **Jednotkové testovanie** overuje vybranú metódu, triedu či menší program vytvorený jedným vývojárom nezávisle na ostatných častiach systému. Jednotkové testovanie je základom pre ďalšie testovania, pretože ak časť kódu nepracuje správne samostatne, nebude použiteľná ani po integrácii do systému.

- **Integračné testovanie** slúži na detekciu chýb v komunikácii (spolupráci) medzi metódami a triedami v rámci komponentu alebo subsystému. Pri integračnom testovaní sa predpokladá, že jednotkové testy a testy komponentov prebehli úspešne (chyba nebola detekovaná).

- **Validácia a overovanie** funkčných požiadaviek je dôležitou súčasťou testovania po ukončení stavby rozhrania (prípadne prototypu).

Testovanie prebieha formou spúšťania testovaného kódu na jeho vstupných dátach, overuje sa výskyt chýb počas vykonávania kódu a porovnávajú sa výstupy s očakávanými hodnotami. Úplné testovanie je testovanie na všetkých vstupných dátach, avšak vo väčšine prípadov je nereálne (možných vstupov je príliš veľa). V praxi sa teda používa neúplné testovanie, ktoré sa zameriava na výber takých typov vstupov, u ktorých je najväčšia pravdepodobnosť, že dôjde ku chybe.

Medzi neúplné testy patrí aj štrukturálny test. „Základná myšlienka spočíva v predpoklade, že každý príkaz v programe musí byť otestovaný aspoň raz. Príkaz

if je logickým príkazom. Príkazy if a while musíte pri testovaní rozlišovať podľa zložitosti riadiacej podmienky. Jedine tak zaistíte, že bude príkaz testovaný zvnútra. Najľahšou pomôckou, ako na nič nezabudnúť, je vypočítať počet ciest programom a následne vyvinúť minimálny počet modelových prípadov, ktorý všetky cesty programom preverí. Minimálny počet modelových prípadov vypočítate jednoduchým spôsobom:

- Za priamy prechod metódou zoberte počiatočnú hodnotu 1.
- Za každé nasledujúce kľúčové slovo if, while, repeat, for, and a or alebo ich obdobu pripočítajte 1.
- Za každú alternatívu prepínača pripočítajte 1. Ak nemá prepínač klauzulu default, pripočítajte ďalšiu 1.

Tento spôsob testovania zaručuje iba skutočnosť, že bude testovaný celý kód, neberie do úvahy obmenu dát.“ [McC04]

Niektoré testy je možné automatizovať a prenechať ich vykonávanie na skripty. Výnimkou je užívateľské rozhranie, ktorého výsledná funkcionalita musí byť otestovaná manuálne. Modularita aplikácie hrá v takomto testovaní dôležitú rolu, nakoľko jednotlivé moduly je možné otestovať ako samostatné celky (či už manuálne alebo automatizovane) a následne riešiť už iba integračné testovanie.

„Ladenie je proces rozpoznávania príčin chyby a následnej opravy týchto príčin.“ [McC04] Na účinné hľadanie chýb je možné použiť nasledujúci postup:

1. Stabilizácia chyby. Nájdite najjednoduchší modelový prípad, ktorý je schopný reprodukovať chybu.
2. Nájdenie zdroja chyby. Vyslovte a preukážte hypotézu o príčinách vzniku chyby pomocou analýzy všetkých nazhromaždených dát (vyvolávajúcich chybu).
3. Oprava chyby.
4. Otestovanie opravy chyby.
5. Pokus o nájdenie ďalších podobných chýb.

Steve McConnell popisuje niekoľko tipov pre hľadanie chýb:

- Skúmanie malých častí kódu pomocou jednotkových testov.
- Reprodukcia chyby rôznymi spôsobmi.
- Zužovanie podozrivej časti kódu.

- Preverenie výskytu chyby v predchádzajúcich verziách kódu.
- Postupná integrácia.
- Správy z prekladačov a interpreterov programovacieho jazyka sú nepresné.
- Rozdeľuj a panuj — rozdelenie kódu na niekoľko častí a hľadanie časti, v ktorej sa chyba prejavuje.

Po odhalení chyby je vhodné samozrejme chybu odstrániť. Pred opravou chyby sa odporúča vykonať zálohu zdrojového kódu pre prípad zlyhania opravy kódu. Po vykonaní opravy je vhodné otestovať nielen opravu chyby ale aj ostatné časti súvisiaceho kódu, vylúčia sa tak vedľajšie efekty opráv.

9.2 Dokumentácia

Dokumentácia projektu je tvorená informáciami v zdrojových kódoch aplikácie a externými dokumentami.
Externé dokumenty sú vhodné na zápis požiadaviek na aplikáciu, návrhu architektúry a manuálu pre užívateľov systému.
Informácie v zdrojových kódoch môžu byť reprezentované formou komentárov alebo dostatočne samopopisného kódu. Dostatočne samopopisný kód je označenie kódu, ktorého telo a účel sú náhodnému čitateľovi ľahko zrozumiteľné. Pri tvorbe samopopisného kódu zohrávajú dôležitú úlohu názvy konštánt, premenných, metód a tried a formátovanie zdrojového kódu.

Komentáre v zdrojových kódoch môžeme rozdeliť do niekoľkých kategórií [McC04]:

- **Opakovanie kódu.** Komentár neprináša žiadnu pridanú informáciu ku kódu, je teda zbytočný.
- **Vysvetlenie kódu.** Komentár objasňuje zložité časti kódu, je užitočný, avšak v mnohých prípadoch je nahraditeľný úpravou kódu na samopopisný kód.
- **Záložka v kóde.** Dočasné informácie v kóde, informujúce napríklad o nedokončených častiach kódu. V rámci vývojárskeho tímu štandardizovať záložky v kóde.
- **Zhrnutie kódu.** Krátke slovné zhrnutie vybraného bloku zdrojového kódu.
- **Objasnenie zámeru kódu.** Popis účelu, pre ktorý bol zdrojový kód vytvorený.
- **Iné informácie.** Autorské práva, čísla verzií, licencie a ďalšie.

V konečnej verzii zdrojových kódov by sa mali vyskytovať iba komentáre spadajúce do niektorej z posledných troch menovaných skupín.

```
function handle_requests()
  {
  db_begin();

  $all_requests_to_handle = db_query("SELECT user_id, post_id, group_id, class_id
                                      FROM student_transfer_requests
                                      ORDER BY request_id ASC");

  while ($request_to_handle = db_fetch_array($all_requests_to_handle))
    {
    $user_id = $request_to_handle['user_id'];
    $group_id = $request_to_handle['group_id'];

    if (group_free_places($group_id) > 0)
      {
      send_transfer_notification_email($user_id, $group_id);
      move_user_to_group($user_id, $group_id);
      }
    else
      {
      /* Find someone from the requested group who wants to swap groups. */
      $all_possible_requests_for_swapping
                    = db_query("SELECT V1.group_id, V2.user_id, T2.post_id
                                FROM visiting_list V1, visiting_list V2, transfer_requests T2
                                WHERE V1.user_id = $user_id AND
                                V2.group_id = $group_id AND V2.user_id = T2.user_id
                                ORDER BY post_id ASC");

      if ($swap_request = db_fetch_array($all_possible_requests_for_swapping))
        {
        send_transfer_notification_email($user_id, $group_id);
        send_transfer_notification_email($swap_request['user_id'], $swap_request['group_id']);

        move_user_to_group($user_id, $group_id);
        move_user_to_group($swap_request['user_id'], $swap_request['group_id']);
        }
      }
    }
  db_commit();
  }
```

Obr. 9.1: Príklad samopopisného kódu funkcie na preradenie študenta do novej skupiny

```
//#1
//vynásobí premenné $a, $b, $c a výsledok uloží do $vysledok
$vysledok = $a * $b * $c;

//#2
//ziska informácie o skupine a odošle na užívateľov a predvolený účet notifikácie o presune užívateľa
$gnameq = db_query("SELECT class_name, group_name, group_time FROM groups NATURAL JOIN classes
 WHERE group_id='$group_id'");
$gname = db_fetch_array($gnameq);
$emailaddrq = db_query("SELECT user_name, user_mail, user_info FROM user_profiles WHERE user_id = '$user_id'");
$emailaddr = db_fetch_array($emailaddrq);
if (trim($emailaddr['user_mail']) != "")
{
  mail(trim($emailaddr['user_mail']."st.fmph.uniba.sk"), "UDDB (".$gname['class_name']."): Presun do skupiny ".
  $gname['group_name'],"Uvod do DB systemov (".$gname['class_name']."): Vas presun do skupiny ".$gname['group_name'].
  " (".$gname['group_time'].") bol pravdepodobne uspesny. Svoj aktualny rozvrh najdete v rezervacnom systeme
  http://cvika.dcs.fmph.uniba.sk/~plachetk/SYSCVIKA/index.php\n");
}

//#3
// ******** TODO - naprogramovat **********

//#4
//funkcia otvorí spojenie s databázovým systémom
function db_connect($host,$dbname,$user,$password,$schema)
  {
  $dbconn = pg_connect("host=$host dbname=$dbname user=$user password=$password");
  if ($schema != "")
    pg_query("SET search_path TO $schema");
  return $dbconn;
  }

//#5
//filtrovanie výstupu a kódovanie špeciálnych znakov
$vys['user_mail'] = filter_var($vys['user_mail'],FILTER_SANITIZE_SPECIAL_CHARS);

//#6
/*This program is free software: you can redistribute it and/or modify it under the terms
of the GNU General Public License as published by the Free Software Foundation, either
version 3 of the License, or (at your option) any later version.*/
```

Obr. 9.2: Príklady všetkých typov komentárov

Záver

Táto práca podáva prehľad základných princípov tvorby interaktívnych webových aplikácií. Presvedčili sme sa, že štandardné nástroje softvérového inžinierstva sú aplikovateľné aj na tvorbu webových aplikácií a že správny návrh architektúry môže mať zásadný vplyv na vývoj aplikácie a jej výslednú podobu. Špecifickým oblastiam tvorby webových aplikácií sme vyhradili niekoľko kapitol, v ktorých sme sa venovali použiteľnému a prístupnému užívateľskému rozhraniu, bezpečnosti a rozumnému využitiu databázového systému ako úložiska dát pre webové aplikácie. Preukázali sme, že aj v procese stavby (implementácie) webovej aplikácie existujú nezávisle na zvolenej technológii nástroje, ktoré zvyšujú zrozumiteľnosť kódu a znižujú náklady na jeho úpravu v budúcnosti. V poslednej kapitole sme uviedli typy testovania aplikácií a dokumentáciu zdrojového kódu pomocou samopopisného kódu a vhodných typov komentárov.

Jednotlivé metódy a nástroje sme demonštrovali na viac ako 43 príkladoch. Príklady boli vybrané z reálne nasadeného systému pre podporu organizácie cvičení z Úvodu do databázových systémov, ktorý vznikol ako ročníkový projekt. Systém pôvodne vznikal bez mnohých znalostí uvedených v tejto práci. Jedným z prínosov tejto práce bolo vylepšenie niektorých častí aplikácie a oprava chýb v zabezpečení aplikácie.

Pre mňa bolo hlavným prínosom tejto bakalárskej práce vytvorenie materiálu pokrývajúceho dôležité časti problematiky tvorby webových aplikácií, ktorý mi v budúcnosti pomôže vyhýbať sa rovnakým chybám, k akým som v dôsledku nevedomosti dospel pri pôvodnej tvorbe systému, a zvýšiť kvalitu a bezpečnosť mojich budúcich webových aplikácií. Oboznámenie sa s vlastnosťami požiadaviek, s metódami ich získavania, s vlastnosťami dobrého návrhu architektúry a s nástrojmi na jeho tvorbu mi umožní do budúcnosti navrhovať dobré architektúry, ktoré budú odolné voči menej zásadným zmenám v požiadavkach a ktoré sa budú dať ľahko a rýchlo implementovať. Získal som tiež prehľad o nástrojoch pre podporu vedenia vývojárskeho tímu a zoznam kritérií pre použiteľné a prístupné užívateľské rozhranie webových aplikácií. Vďaka oboznámeniu sa s problematikou bezpečnosti webových aplikácií (a ich komunikácie s užívateľom) prostredníctvom tejto práce som si rozšíril poznatky získané v rámci môjho štúdia na fakulte a

s ich prispením budem schopný už pri tvorbe ďalších aplikácií implementovať ochranu proti mnohým útokom na webové aplikácie.

Verím, že práca bude rovnakým spôsobom prínosná aj pre čitateľa a aj na jej základe vytvorí svoje aplikácie efektívnejšie, kvalitnejšie a bezpečnejšie.

Literatúra

[Cod70] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970.

[Dij72] Edsger W. Dijkstra. The humble programmer. *Commun. ACM*, 15(10):859–866, 1972.

[EN03] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems, Fourth Edition.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[Hil96] Carol Hildenbrand. Loud and clear. *CIO Magazine*, April 1996.

[Hoa81] Charles Antony Richard Hoare. The emperor's old clothes. *Commun. ACM*, 24(2):75–83, 1981.

[HT99] Andrew Hunt and David Thomas. *The pragmatic programmer: from journeyman to master.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[Hus04] Sverre H. Huseby. *Innocent Code: A Security Wake-up Call for Web Programmers.* John Wiley & Sons, Hoboken, NJ, USA, 2004.

[Kri92] S. Krishna. *Introduction to database and knowledge-base systems.* World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1992.

[McC04] Steve McConnell. *Code Complete, Second Edition.* Microsoft Press, Redmond, WA, USA, 2004.

[Nie] Jakob Nielsen. Jacob Nielsen's Alertbox. `http://www.useit.com/alertbox`.

[OWA09] OWASP. Command injection, March 2009. `http://www.owasp.org/index.php/Command_Injection`.

[Pec07] Rudolf Pecinovský. *Návrhové vzory.* Computer Press, a. s., Brno, Česká republika, 2007.

[Sch05] Kathy Schwalbe. *Information Technology Project Management, Fourth Edition.* Course Technology, 2005.

[UW01] Jeffrey D. Ullman and Jennifer Widom. *A First Course in Database Systems.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[Wie99] Karl Eugene Wiegers. *Software Requirements.* Microsoft Press, Redmond, WA, USA, 1999.