VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

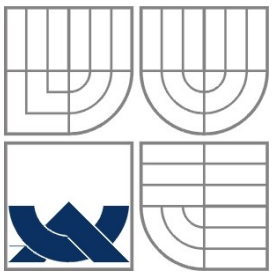
FRAMEWORK PRO VÝVOJ ADMINISTRAČNÍCH SYSTÉMŮ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

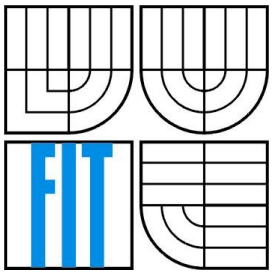
AUTOR PRÁCE
AUTHOR

BC. MAREK ČEVELÍČEK

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

FRAMEWORK PRO VÝVOJ ADMINISTRAČNÍCH SYSTÉMŮ

FRAMEWORK OF ADMINISTRATION SYSTEMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

BC. MAREK ČEVELÍČEK

VEDOUCÍ PRÁCE
SUPERVISOR

ING. VLADIMÍR BARTÍK, PH.D

BRNO 2008

Abstrakt

Projekt pojednává převážně o teoretické části tvorby on-line provozovaných administračních systémů, nastiňuje výhody frameworků a zabývá se návrhem uživatelského rozhraní. Jsou shrnuty obecné principy tvorby interaktivních systémů v jazyce (X)HTML s využitím formulářových prvků, JavaScriptu či AJAXu. Zvláštní ohled je brán na bezpečnostní mechanismy pro optimální zabezpečení aplikací v PHP.

Klíčová slova

framework, informační, administrační, systém, PHP, MySQL, on-line, modul, generátor, bezpečnost, HTTP, HTTPS, sessions, přístupnost, použitelnost, XHTML,

Abstract

Project deals on theoretical part of the process of developing administration systems which are supposed to run on-line. It talks about advantages of frameworks and finds basic principles of designing user interfaces. There are summarized general principles of using interactive components as (X)HTML form elements, JavaScript and AJAX with respect to modern administration systems. Important part of this work describes use of an optimal security mechanisms in language PHP.

Keywords

framework, information, administration, systém, PHP, MySQL, on-line, module, generator, security, HTTP, HTTPS, sessions, accessability, usability, XHTML

Citace

Čevelíček Marek: Framework pro vývoj administračních systémů. Brno, 2008, semestrální projekt, FIT VUT v Brně.

Framework pro vývoj administračních systémů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením
Ing. Vladimíra Bartíka, Ph.D
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jméno Příjmení, Datum

Obsah

Obsah....1
1 Úvod....3
2 Administrační systémy....5
2.1 Uživatelské rozhraní....6
2.1.1 Formuláře....8
2.1.2 JavaScript a AJAX....9
2.2 Bezpečnost a zebezpečení aplikace....10
2.3 Autentifikace uživatele....13
2.3.1 Autentifikace pomocí HTTP....13
2.3.2 Protokol HTTPS....13
2.3.3 Sessions....14
2.3.4 Metoda Challenge-Response....14
2.4 Všeobecné postupy při návrhu databáze....15
2.5 Framework....15
2.5.1 Vzájemně provázané knihovny....16
2.5.2 Samostatně použitelné knihovny....16
2.6 Architektonické vzory....16
2.6.1 Model-1....17
2.6.2 Model-View-Controller (Model-2)....17
3 Administrační framework Liquid Admin....19
3.1 Požadavky frameworku....20
3.1.1 Funkční....20
3.1.2 Datové....21
3.2 Architektura aplikace....22
3.2.1 Návrhové vzory tříd....22
3.2.2 Struktura modulu....23
3.2.3 Integrace modulu....24
3.2.4 Bezpečnostní prvky....25
3.3 Implementované principy....29
3.3.1 Autentifikace uživatele....29
3.3.2 Autorizace uživatele....31
3.3.3 Navigace v modulech....32
3.3.4 Logování informací....33
3.4 Generátor modulů....34

4 Vzorový administrační systém 37
4.1 Návrh systému 37
4.2 Systémové prvky 38
4.3 Moduly systému 39
4.3.1 Modul zákazníci (customers) 39
4.3.2 Moduly kontakty (contacts) a projekty (projects) 41
4.3.3 Zavedení modulů 41
4.4 Instalace a testování 42
5 Nasazení a provoz administračních systémů na frameworku Liquid Admin 43
5.1 Provozní prostředí 43
5.2 Zhodnocení provozu – Megacars.cz 43
5.3 Zhodnocení provozu – dokumentová knihovna, http://aukcevozidel.cz/doclibrary/ 45
6 Pokračování projektu 47
7 Závěr 48
7.1 Přínos projektu 48
Literatura 50
Seznam příloh 51
Příloha 1 – Knihovny frameworku 52
Příloha 2 – SQL tabulek knihovny Auth 61

1 Úvod

Těžko byste hledali člověka, který neslyšel nikdy slovo "internet". Různí lidé si pod tímto pojmem představují různé věci. Především je však slovo internet spjato se službou World Wide Web, která pomohla zásadním způsobem k jeho masovému rozšíření. Existují stránky, které lidé navštěvují a dovídají se na nich potřebné informace, hledají zábavu či komunikují s přáteli. Internet pronikl do mnoha částí lidského života a dnes se dá považovat za stejnou samozřejmost, jako televize či rádio, avšak s mnohem variabilnějšími možnostmi využití.

Stejně rychle, jako internet se rozrostla skupina společností, které vytvářejí nové služby a zpřístupňují je lidem. Komunitní uskupení osob pracují na projektech, aby ostatním uživatelům poskytli nástroje pro kvalitnější práci nebo lepší a rok od roku propracovanější zábavní systémy.

Z internetu se stal rovněž prostor pro podnikání. Takové společnosti spojuje jediné slovo - reklama. Nespočet firem je schopno vyvíjet různé aplikace pro lepší prezentaci produktů, cílené reklamní kampaně, poskytovat analýzy trhu a požadavků zákazníků. Všechny tyto služby jsou vyhledávané a rovněž dobře placené, jelikož vyžaduje mnohé úsilí zorientovat se v tolika směrech, kterými se internet dokázal vyvinout. Pracovníci těchto firem musejí strávit mnoho let studiem a projít několika školeními, aby pochopili souvislosti internetové reklamy a ani ti nejlepší nebudou nejspíš nikdy znalí ve všech směrech internetového podnikání.

Není však nutné dále ze široka rozebírat tuto problematiku. Jak již jsem nastínil výše, hlavní a zásadní vliv na rozvoj internetu měla a nadále má služba World Wide Web (nebo-li lépe známý pojem - webové stránky) a kolem tohoto pojmu se bude točit i moje diplomová práce. Webové stránky mají různý účel a daly by se kategorizovat rozličnými metodami. Spojuje je však jedna věc - prezentace informací a s ní úzce spjatý problém, jak tyto informace spravovat.

Kde jsou informace uloženy je většinou na okraji zájmu koncového zákazníka. Důležité je, aby datový sklad splňoval potřebné podmínky pro bezpečnost, dostupnost a aktuálnost. Bezpečnost datového skladu ve valné většině určuje projektant systému a dostupnost informací je předmětem objednávek. K tomu celý cíl směřuje - aby informace, které máme, mohly být rozšířeny zákazníkům v dostatečné míře. Podstatnou podmínkou šířených informací však je, aby byly aktuální, protože neaktuální informace mají nulovou hodnotu. A kdo má nad aktuálností informací větší přehled, než samotný poskytovatel? Snaha přenést manipulaci s datovým skladem na poskytovatele informací je tedy primárním cílem vytváření informačně orientovaných webových projektů. Snaha o vytvoření nástroje, který by mohl i běžný uživatel internetu prezentovat své informace bez nutnosti znát programovací jazyk a jiné složité techniky, které jsou v souvislosti s webovými systémy.

Jednoduchost je trend vývoje moderního software. K čemu je dobré, že na obrazovce monitoru vidíte nespočet graficky přehnaných oken s různými formuláři a relevantními informacemi, když v záplavě všech těchto objektů stěží dokážete vyjádřit, co je důležité sdělit? O prezentaci informací

bylo napsáno mnoho knih stejně jako o vývoji robustních administračních systémů. Tato knihy jsou zajisté velmi kvalitní a obsahují množství rad, postupů a demonstrací, jak vytvořit to či ono. Málokteré knihy se však zabývají do šíře souvislostmi programovacích technik a uživatelským prostředím. Přesně to je cílem této diplomové práce - spojit návrh administračního systému a uživatelského rozhraní tak, aby výsledkem byl framework, v němž nebude nutné hledat v záplavě komponent. Primárním účelem je vytvořit framework, který bude dobře použitelný pro většinu **středně velkých** projektů, bude obsahovat potřebné komponenty a soubor návodů, jak tyto komponenty používat, aby výsledný administrační systém byl efektivně použitelný.

Několik let jsem komunikoval s různými lidmi a zjišťoval, co je pro ně důležité. Ptal jsem se na jejich požadavky k prezentaci informací a poslouchal, co upřednostňují. Snažil jsem se navrhovat různé varianty prezentací a obměny těchto variant. Různí lidé chtěli různé věci, někteří dokonce nevěděli, co chtějí, ale po dlouhých debatách vyšlo najevo, že mají jedno společné - chtějí přehlednost systému a hlavně jednoduchost správy. Až na dalších místech se zobrazovaly body, jako grafická originalita a speciální funkce.

Jako programátor jsem zároveň schopen představit si, co se bude skrývat v pozadí aplikací, které tito lidé požadují a vybrat vhodnou technologii, analyzovat strukturu systémů a vytvořit jakousi univerzální kostru.

Je to složité a zdlouhavé studium a věřím, že budu moci v budoucnu napsat mnoho inovovaných informací o vývoji aplikací. Nyní se Vám pokusím podat vědomosti, které mám a touto prací ucelit své poznatky z konstruování administračních systémů. Produktem celého snažení bude administrační framework, pomocí něhož bude možné konstruovat bezpečné a uživatelsky příjemné administrační systémy k webovým projektům.

2 Administrační systémy

Vývoj systému začíná vždy úvahou nad tím, kdo bude systém používat a k jakému účelu bude soužit. Administrační systémy, jak již název napovídá, slouží k administraci (tedy správě) buď celé softwarové aplikace nebo datového skladu.

Webová aplikace je podle definice klient/server software, který komunikuje s uživatelem nebo jiným systémem prostřednictvím protokolu HTTP. Za klienta používají uživatelé nejčastěji webové prohlížeče, jako je Internet Explorer, Firefox nebo Opera. Automatizované systémy, například roboti fulltextových vyhledávačů, pak přistupují s pomocí HTTP agentů. Klient na základě interakce s uživatelem zasílá serveru jednotlivé požadavky a následně zobrazuje obdržené webové stránky zapsané zpravidla v jazyce HTML. Rozsah aplikací poskytovaných služeb se může pohybovat od jednoduchých úloh, jakými je například kniha hostů, až po sofistikované programy typu komplexních on-line obchodů či bankovních aplikací.

Administrační systém webové aplikace bývá většinou umístěn on-line, stejně, jako samotná aplikace. Výjimkou je administrace umístěná na jiném serveru. Je-li administrace přístupná on-line, nese to s sebou výhodu v tom, že správce může ovládat systém z jakéhokoliv počítače připojeného k internetu. Nevýhoda je naopak v nutnosti co nejlépe zabezpečit takový administrační systém. Je totiž mnohem snáze napadnutelný, než systém umístěný na chráněném serveru, kam mohou přistoupit pouze někteří uživatelé. O bezpečnosti a hrozbách pro on-line systémy pojednává dále kapitola **Bezpečnost a zabezpečení aplikace**.

Mluvíme-li o administraci webových projektů, pak bývá zvykem, že administrační systém bývá naprogramován ve stejném jazyce, jako prezentační vrstva softwarové aplikace. Výjimečně se lze setkat s administračním systémem například v jazyce Java, zatímco aplikace samotná je naprogramována v PHP. Toto bývá výsadou snad jen velkých projektů, kde výběr jiného programovacího jazyka měl vliv na některou další stránku fungování aplikace, jako bezpečnost, či snadnější vývoj a správa software.

Administrační systémy, které jsou předmětem této práce budou sloužit ke správě webových projektů. V současné době jsou asi 4 nejrozšířenější webové programovací jazyky - ASP.NET, PHP, Java a v poslední době popularizovaný jazyk Ruby na platformě Rails (Ruby on Rails - RoR). Srovnání jazyků a technologií je velmi složité a někdy ne zcela možné. Každý má své výhody a nevýhody a některé jazyky jsou propagovány jako celá platforma či framework (ASP.NET, RoR), zatímco jiné existují pouze jako programovací jazyk a každý musí pro svou potřebu vytvořit soubor použitelných knihoven, aby v něm bylo možné pracovat efektivně.

Moje práce se bude zabývat realizací frameworku v jazyce PHP. Tento poměrně mladý programovací jazyk doznal v postupu let mnoha změn. PHP 3 byla první použitelná verze, která se velmi rychle rozšířila. Přesto obsahovala množství nedostatků, které byly pro použití

v profesionálních aplikacích nepřekonatelnou překážkou. Verze 4, jedna z nejoblíbenějších verzí, již skutečně mohla být používána i pro vývoj velkých a kvalitních systémů. Mnoho vývojářů v ní však postrádalo kvalitní objektový model, který měl přijít až v PHP 5. Záhy se ukázalo, že ani pátá verze neimplementuje všechny potřebné objektové rysy. Přesto byla dokonalejší a použitelnější než všechny předchozí verze.

Objektové programování přináší do aplikace větší míru přehlednosti a rozšiřitelnosti. Větší projekty, které jsou psány strukturovaně a za pomocí ohromného počtu funkcí, se stávají po mnohých úpravách a opravách nepřehledné natolik, že je nutné takto vytvořené aplikace kompletně přepisovat. To je beze sporu náročná práce. Pro vývoj frameworku jsem vybral verzi PHP 5.

Dnes již je snahou vývojářů spíše podporovat PHP verzi 5 a zatlačit verzi 4 do ústraní. Důvodem je hlavně částečná nekompatibilita těchto dvou verzí. Každá pracuje nepatrně jiným způsoben například s řetězci nebo objekty a je pravdou, že PHP 5 je v celkovém ohledu propracovanější. Poskytovatelé internetových služeb, zvláště pak webhostingů mají spoustu práce s přípravou dvou různých verzí tohoto jazyka.

Na internetu vznikla dokonce komunita, která nese název GO PHP 5 (www.gophp5.org) a její snahou je podpořit přechod vývojářů na novější verzi jazyka. Na svých stránkách sdružují seznam projektů, které již přešli na novou verzi a namísto udržování kompatibility s verzí 4 se snaží o zlepšení objektového návrhu, který je v PHP 5 lépe realizovatelný a působí ucelenějším dojmem.

2.1 Uživatelské rozhraní

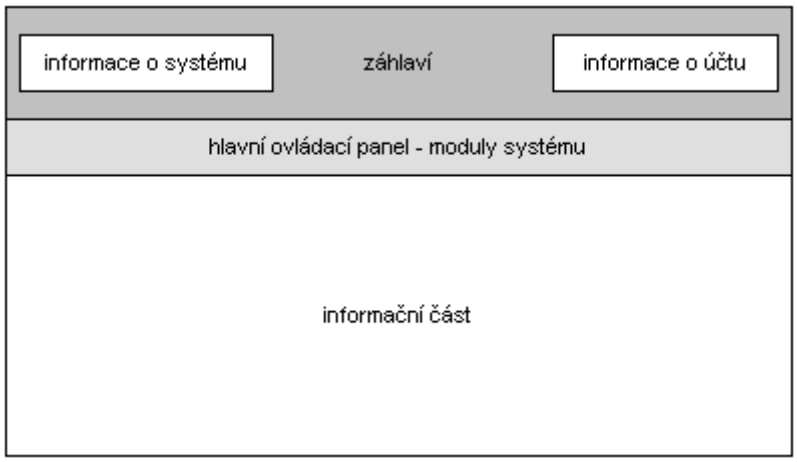
Uživatelské rozhraní je jednou z nejdůležitějších částí každé aplikace. Je to ta část, kterou uživatel vidí, kterou řídí aplikaci. Budeme-li o uživatelských rozhraních mluvit jako o webových administračních systémech, pak jsou zobrazována jako HTML stránky se všemi náležitostmi. Klientským systémem, na němž aplikace běží, je internetový prohlížeč a interakce aplikace a uživatele je zajištěna prostřednictvím formulářových prvků jazyka HTML.

Layout, tedy uskupení prvků na obrazovce, je možné rozdělit do několika částí. Každý administrační systém by měl obsahovat jisté nezbytné informace, které mají z hlediska přehledností zaběhnuté konvence umístění. Jedná se především o:

- informace o systému (záhlaví)
- informace o účtu uživatele (záhlaví)
- hlavní ovládací panel
- informační část

Po přihlášení do administračního systému by měl uživatel bezpečně vědět pod jakým účtem je přihlášen. Titulní obrazovka administračního systému by měla obsahovat informace co je

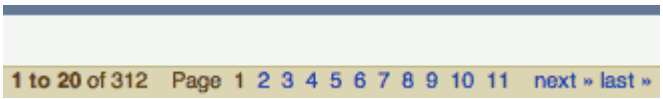
to za systém, co jím lze ovládat a kde se právě uživatel nachází. Je vhodné rovněž uvést i další informace, jako kdy byl uživatel naposledy přihlášen a z jaké IP adresy do administrace přistoupil. Jedno z možných rozvržení layoutu je zobrazeno zde:



Obrázek 1 - Layout administračního systému

Uživatelské rozhraní aplikace bude sestávat z informací o přihlášeném uživateli, vhodně umístěného odkazu k odhlášení, hlavního ovládacího panelu, který bude odkazovat na moduly administračního systému a poslední částí bude informační část. Informační část je nejdůležitějším prvkem layoutu. Je to část, kde většina programátorů vytvoří velké množství chyb v souvislosti s přehledností systému.

Informační část obsahuje data seřazená definovaným způsobem většinou v tabulce, kde každý řádek tabulky odpovídá jednomu datovému záznamu. S datovými záznamy je možné provádět běžné akce, jako editace, smazání, či prohlížení detailu (datové záznamy mohou být totiž někdy značně obsáhlé a zobrazení v plné formě znepřehledňuje celkový pohled na data). Data v databázích bývají mnohdy mezi sebou propojena a je bežné, že každý datový záznam obsahuje i odkaz na zobrazení souvisejících záznamů. Je-li datových dádnamů vetší množství, pak je vhodné implementovat i nějakou formu postupného zobrazení záznamů. Nejčastěji jsou tedy záznamy rozděleny po určeném množství na několik stran a aplikace zahrnuje ovládání, které umožňuje prohlížet tyto strany. Stránkování datových záznamů s sebou nese ještě další problém, který se projeví, když se počet stran se záznamy vyšplhá na stovky či tisíce, jak tomu bývá například v administračních systémech diskuzních fór. Zde je vhodné implementovat některou z pokročilých stránkovacích technik, jako například zobrazení série deseti stran s odkazy na odskok na předchozí a další sérii, jak ilustruje následující obrázek.



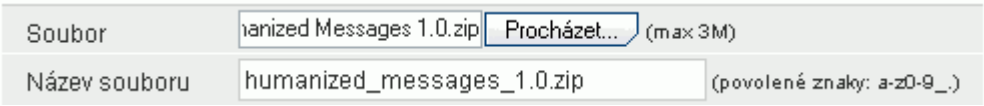
Obrázek 2 - Stránkování v monitorovacím nástroji databáze MySQL

Při vysokém počtu datových záznamů je rovněž vhodné implementovat i nějaký způsob vyhledávání v záznamech, nebo alespoň filtrování záznamů, aby uživatel nemusel listovat celým seznamem.

2.1.1 Formuláře

Hlavním prvkem uživatelského rozhraní budou také formuláře, pomocí nichž bude uživatel do systému vkládat nová data a upravovat stávající. Při návrhu struktury formulářů bylo definováno tolik obecných pravidel, že by to stačilo na vydání celé jedné knihy. Zde se pokusím pouze zdůraznit některá zásadní.

V první řadě je nutné si uvědomit, že klientská část administrace poběží v internetovém prohlížeči jako HTML stránka, a tedy máme na výběr pouze omezené množství komponent, jenž HTML jazyk definuje. Každá komponenta má své jedinečné využití a je nutné zvolit pro patřičné objekty vhodné komponenty. V jazyce JAVA a podobných je možné komponenty rozličným způsobem upravovat a formovat tak, aby dělaly to, co potřebujeme. Budeme-li chtít dosáhnout podobného efektu v tomto případě, pak bude nutné sáhnout po nějakém skriptovacím jazyku. Volba padne nejpravděpodobněji na JavaScript. Stačí však, aby měl uživatel ve svém internetovém prohlížeči podporu JavaScriptu vypnutou a celé snažení o přestavbu formulářových komponent vyjde naprázdno. Formuláře jsou jediným pojítkem uživatele s datovým úložištěm. Je kriticky důležité zajistit, aby bylo umožněno uživateli manipulovat s daty, s nimiž manipulovat chce (a má k tomu samozřejmě požadovaná oprávnění).



Obrázek 3 - Název souboru vygenerován JavaScriptem z originálního názvu souboru na disku, aby neobsahoval nepovolené znaky

Tvorba dobrého formuláře se drží několika pravidly:

Používání správných komponent pro určené volby. Pro textové vstupy se použije komponenta **input** nebo **textarea**. Je-li na výběr více možností, pak lze použít komponenty **radio**, **checkbox**, **selectbox**. S výhodou použijeme dobře strukturovaný selectbox v místech, kde by vypisování voleb jako radio butony zabralo příliš mnoho místa na stránce.

Vhodné zarovnání komponent na stránce, oddělení logických celků pomocí tagu **fieldset** a pojmenování návěstí pomocí **label**. Pomocí kaskádových stylů lze formuláře velmi dobře uspořádat. Internetové prohlížeče již dnes poměrně slušně interpretují většinu prvků jazyka CSS.

Čitelný styl. Je prioritou, aby se v designu neztrácel kontext. Uživatel musí jasně vědět k které pole formuláře edituje a toto pole musí být viditelné. Bohužel každý prohlížeč si s formuláři poradí podle svého a je to velká nevýhoda. Safari zobrazí formulářové prvky jinak, než Firefox - možná je účelem vývojářů zamezit přehnané stylování formulářů tím, že zobrazí jednotný vzhled veškerých formulářových prvků. Každopádně již mnohokrát bylo napsáno v tutoriálech na internetových magazínech, že je lépe ponechat nativní podobu formulářů a opominout jakékoliv stylování, než se pak dočkat špatného výsledku ze strany zobrazení formulářů.

2.1.2 JavaScript a AJAX

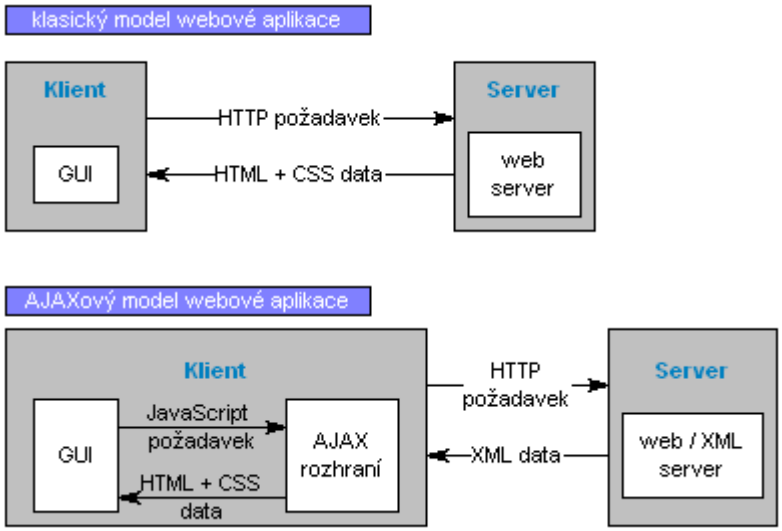
Mnohdy je pohodlné provést některé akce na klientské straně aplikace před odesláním dotazu na server. V takových případech je dobré použít JavaScript. Hodí se pro programování drobných a práci ulehčujících funkcí. Pár projektů se pokusilo použít JavaScript i pro tvorbu jednotného webového frameworku (Scriptaculous, Prototype) či pro desktopové aplikace (Yahoo Widgets). Ačkoliv mnoho projektů založených na JavaScriptu je úspěšných, jedná se většinou o vylepšení prezentační části systému, nikoliv administrace. Při používání tohoto jazyka v administračních systémech je však doporučeno držet fantazii zkrátka. Snadno lze naprogramovat zdánlivě elegantní aplikaci, která s sebou přináší skrytá bezpečnostní rizika. Některé JavaScriptové funkce nelze ani efektivně implementovat, aniž by mezi sebou nevytvářely kolize při používání různých internetových prohlížečů, což je u většiny projektů nežádoucí. Veškeré snahy vývojářů o přístupný software totiž směřují k tvorbě aplikací, které budou provozu schopné na běžném vybavení uživatele. Použitelnost software určuje, jak snadno se na nich uživatelé orientují, jak rychle pochopí jejich uspořádání a ovládání a jaký uživatelský zážitek si z nich odnesou.

JavaScriptová technologie může být při uváženém použití velmi dobrým pomocníkem. Například potvrzení žádosti o smazání záznamu (při případné chybě uživatele) může využít událost, která se provede při kliknutí na ovládací prvek místo složitého generování dotazu na server a zpracovávání odpovědi. Takto lze ulehčit režii serveru a zlepšit rychlost a pružnost ovládání aplikace.

Programátor by neměl spoléhat se na podporu JavaScriptu, pokud jde o systémy, v nichž hraje velkou roli bezpečnost a integrita informací. JavaScript lze použít k usnadnění práce, avšak funkcionalita aplikace musí být zachována i při jeho absenci.

S tématem skriptovacích jazyků úzce souvisí AJAX (Asynchronous JavaScript and XML). Je to obecné označení pro technologie vývoje interaktivních webových aplikací, které mění svůj obsah bez nutnosti jejich kompletního znovu-načítání. Na rozdíl od klasických webových aplikací poskytují uživatelsky příjemnější prostředí, ale vyžadují použití moderních webových prohlížečů. Někteří programátoři začali s příchodem technologie AJAX implementovat do administračních systémů ajaxové prvky. Mým názorem je být při výběru těchto prvků velmi opatrný. Technologie AJAX

funguje na bázi jazyka JavaScript a platí pro ni tedy stejná pravidla použití a stejná omezení. Špatně navržené ajaxové prvky mohou vážně ohrozit bezpečnost celého systému.



Obrázek 4 - Klasický a AJAXový model webové aplikace

2.2 Bezpečnost a zebezpečení aplikace

Na úvodu této kapitoly bych rád definoval pojmy bezpečnost a zabezpečení. Od jejich významu se bude dále odvíjet pojednání následujícího textu.

Bezpečnost aplikace se dotýká vnitřních i vnějších mechanismů a vlastností aplikace, které pomáhají chránit zájmy všech uživatelů a prvky celého systému. Patří sem tedy systémové prvky, ale i role lidského faktoru. Zabezpečení zahrnuje principy, které chrání systém před vniknutím neautorizovaných osob a brání jeho napadení se záměrem poškodit data nebo samotný systém.

Aplikace bude pouze tak bezpečná, jak bezpečně ji budou používat její uživatelé. Není v lidských silách eliminovat veškerá rizika, která s sebou provoz on-line administračních systémů přináší. Byly ovšem vynalezeny mnohé principy a postupy, jak minimalizovat riziko zneužití či vyzrazení citlivých informací. Při aplikování těchto principů je vždy nutno uvažovat míru ochrany požadované aplikace. Systémy zdánlivě podobné mohou spadat do naprosto odlišných kategorií důvěryhodnosti a utajení. Pro bankovní systémy není možné použít jednoduché bezpečnostní prvky a nopak pro jednoúčelové administrační systémy není nutné používat předimenzované ochranné prostředky.

Bezpečnost i zabezpečení se lépe chápou jako veličina, kterou je možné měřit. Stoprocentní bezpečnosti nelze nikdy dosáhnout. Jedno programátorské pravidlo říká, že v každém systému je minimálně jedna chyba. Jedna chyba může v některých případech znamenat kritický bod, který může vyústit v totální selhání systému. Těžko budeme hledat naprosto bezpečný systém. Pro konkrétní případ se vždy musí individuálně posoudit, jakou míru bezpečnosti bude systém vyžadovat. Rozhodnutí závisí především na tom, s jakými daty bude systém pracovat. Dnes se již běžně

používají šifrované přenosy citlivých dat i propracované mechanismy ověření identity uživatele (autentifikace).

Pro účel této práce je důležitá bezpečnost samotné aplikační vrstvy. V tomto kontextu existují všeobecná doporučení, která by měla aplikace splňovat.

1/ Validace vstupů a výstupů

Vstupy a výstupy jsou nejčastějším cílem útoků. Vstupní parametry představují místo, kudy lze vniknout do interních struktur systému a na nezabezpečeném systému je tak provést nepovolené akce. Na principu neošetřených vstupních a výstupních parametrů fungují i SQL injekce, které představují významné procento všech útoků na internetové aplikace.

2/ Bezpečné selhání aplikace

Může se stát, že v důsledku vnitřních kolizí serveru či systému hrozí jeho selhání. V takovém případě by vše mělo být připraveno tak, aby nedošlo ke kolizím s předpokládaným chováním. Databáze musí zůstat konzistentní a žádný uživatel by neměl získat větší možnosti, než které jsou mu přiděleny. Bezpečné selhání sice zabrání uživatelům využívat některých částí aplikace, ale neohrozí data.

3/ Jednoduchý bezpečnostní mechanismus

Mluvíme-li o přístupných systémech, pak jednoduchost má vliv na atraktivnost aplikace. Budou-li uživatelské akce složité, systém nebude tak příjemný a možná jej uživatelé zcela odmítnou používat. Uživatelské akce by z pohledu uživatele měly být co nejjednodušší.

4/ Plánování neočekávaných událostí

Někdy není možné předpokládat výskyt nějaké chyby, ale i takové věci se stávají. Je-li systém závislý na několika dalších komponentách či jiných systémech, neměli bychom spoléhat na jejich stoprocentní funkčnost. Je nutné předvídat řešení při nefunkčnosti těchto částí a zajistit, aby nedošlo k poškození a havárii našeho systému. Systém bude pouze tak bezpečný, jako jeho nejslabší článek.

Správné vstupy a výstupy aplikace jsou přes všechny ostatní doporučení nejdůležitější. Způsob, jakým se provádí ověření správnosti blízce souvisí s architekturou aplikaca. Většina aplikací má implementován jednotný způsob kontroly. Většinou je k tomu vyhrazena samostatná komponenta, která provádí veškerou kontrolu a nadřazené komponenty informuje o (ne)úspěchu. Řízení vstupně výstupních parametrů lze provést dvěma způsoby.

1/ Povolení pouze validních a očekávaných parametrů

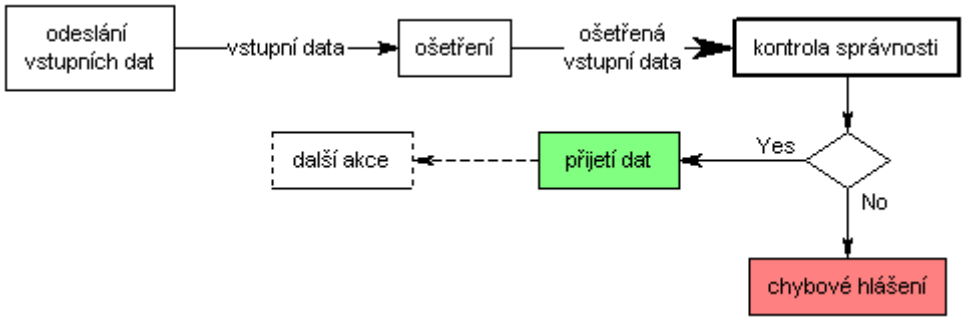
Což vlastně znamená, že pokud bude na vstupu některý parametr v nesprávném tvaru, bude vstup odmítnut. Většinou se odmítne vstup jako celek, pokud alespoň jeden z parametrů nevyhovuje

požadavkům. Dále může být vstup odmítnut také pokud se v něm budou vyskytovat parametry, které jsou navíc, v programu se s nimi nepočítá a jsou tedy zbytečné. Záleží na tom, jakou úroveň bezpečnosti implementuje aplikace.

2/ Zamítnutí chybných parametrů

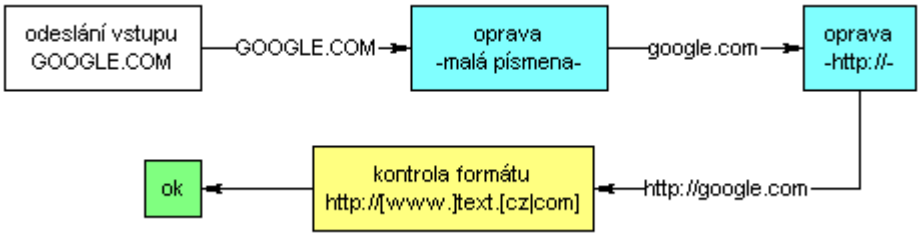
Tímto v programu vytvoříme prostředí, které zpracovává všechny vstupní parametry a odmítá pouze ty z nich, které nevyhovují našim požadavkům. Toto je velmi často používáno v běžných webových aplikacích. Jde o případ, kdy například očekáváme, že parametr ID bude obsahovat pouze číslo a na vstupu dostaneme textový řetězec, který by mohl obsahovat kód pro SQL injekci.

Pokročilejším způsobem je přidání opravné funkce do kontrolní komponenty. Komponenta se nejprve pokusí opravit vstupní parametry a teprve poté kontroluje, zda vyhovují zvolenému kritériu. Při opravování je však důležité vědět, které chybné stavy by se mohly v parametrech vyskytovat. Příkladem je například očekávání webové adresy ve formátu http://www.neco.cz, zatímco na vstup přijde řetězec www.neco.cz. Víme, že na počátku by se mělo v každém případě vyskytovat http:// a můžeme si tedy dovolit přidat jej do vstupu, pokud v něm chybí. Opravování nemusí být stoprocentně dokonale technicky vyřešené - jde pouze o podání pomocné ruky uživatelům. Namísto toho, abychom zobrazili chybovou hlášku ihned po obdržení špatného vstupu, pokusíme se data opravit a teprve pokud ani po kontrole nebudou vyhovovat, zobrazí se uživateli chybové hlášení.



Obrázek 5 - Prověření vstupních dat

Při opravování vstupních dat může být na vstupní parametr aplikováno libovolné množství opravných filtrů. Následující obrázek demonstruje dvakrát opravený vstupní řetězec.



Obrázek 6 - Opravení vstupního řetězce GOOGLE.COM a následné přijetí

2.3 Autentifikace uživatele

2.3.1 Autentifikace pomocí HTTP

PHP podporuje standardní autentifikaci pomocí protokolu HTTP pokud je nainstalováno jako modul Apache. Použití tohoto způsobu autentifikace je poměrně jednoduché. Stačí do prohlížeče zaslat následující dvě hlavičky, které způsobí vyvolání přihlašovacího okna.

```
header('WWW-Authenticate: Basic realm="My Realm"');
header('HTTP/1.0 401 Unauthorized');
```

Takový způsob přihlašování má však některé nevýhody. HTTP je bezstavový protokol, a proto jsou přihlašovací údaje při každém požadavku na provedení akce odesílány znovu na server v nezakódované podobě. Mohou tedy být snadno odchyceny a zneužity cizími osobami. Dalším nedostatkem je nesnadné odhlášení uživatele ze systému. Dnešní prohlížeče se chovají podivným způsobem a každý má své odlišnosti. Univerzální metoda na odhlášení uživatele neexistuje. Bylo vymyšleno mnoho způsobů, které částečně obcházejí tento nedostatek, ovšem ani na ně nemůže být stoprocentní spolehnutí. Oblíbená metoda je například použití dočasné cookie (po dobu trvání session), která bude signalizovat, zda je uživatel odhlášen. Fatální nevýhoda tohoto způsobu spočívá v tom, že zkušený uživatel může zaslanou cookie jednoduše smazat, a tak získat opět přístup do systému, jako by se vůbec neodhlásil. Jistě si dokážete představit, jak snadno lze provést útok například na veřejném počítači, u nějž se střídá více lidí.

Jediná spolehlivá metoda odhlášení je tedy uzavření okna prohlížeče. Tak jej donutíme, aby zapomněl přihlašovací údaje v HTTP hlavičce. Je to však poněkud nepohodlný způsob odhlašování. I tato metoda ztrácí svůj smysl, pokud bylo z okna, které obsahuje přihlášeného uživatele, otevřeno nové okno (potomek). Toto nové okno totiž mnohdy zdědí hlavičku rodiče a přijme ji za svou vlastní. Bude tedy obsahovat i uživatelem odeslané přihlašovací údaje k systému.

2.3.2 Protokol HTTPS

Mnohem sofistikovanější je metoda přihlašování uživatelů pomocí protokolu HTTPS. Funguje na stejném principu jako HTTP, ovšem všechna přenášená data jsou šifrována za pomocí klientského certifikátu. Tento způsob přihlašování se používá často i v on-line bankovnictví.

Bylo by ovšem mylné domnívat se, že tento způsob přihlašování je nenapadnutelný. Útočník v tomto případě nikdy nezjistí přístupové údaje uživatele. Pro získání přístupu do takto zabezpečeného systému není přitom tato znalost nutná. Stačí, když se útočník dostane k zakódovaným datům. Při použití pokročilých technik útoku a s využitím stejného klientského

certifikátu lze získat plnohodnotný přístup do systému. Protokol HTTPS lze však považovat za jeden z nejbezpečnějších způsobů přihlašování uživatelů díky velké složitosti jeho napadení.

2.3.3 Sessions

Přihlašování uživatelů, které je řešeno pomocí sessions dovoluje zajistit velmi vysokou bezpečnost. Lze si vybrat z množství technik, které se uplatní při přihlašování do takto navrženého systému. Nevýhodou je větší náročnost realizace těchto přihlašovacích technik, která vyžaduje rozsáhlé znalosti z této oblasti. Jazyk PHP obsahuje mnoho záludností, které mohou nezkušeného programátora zmást.

Obecný princip této techniky spočívá v ověření totožnosti uživatele a vygenerování tzv. SESSION ID, které se nadále používá pro kontrolu autorizace uživatele. Ve starších verzích PHP se vyskytovalo několik chyb, které umožňovaly potenciálním útočníkům provádět pokročilé útoky na odcizení SESSION ID. PHP verze 5 je o mnoho odolnější, ovšem stále existují nepříliš známé a rozšířené techniky, pomocí nichž lze odcizit informace o přihlášených uživatelích. Některé typy útoků využívají špatné nastavení nebo zabezpečení serverů, jiné přímo útočí na účet přihlášeného uživatele. Vysoké procento útoků lze odrazit, pokud do aplikace implementujeme kontrolu IP adresy přihlášeného uživatele. Jak už to bývá, i tato obrana má svou nevýhodu – uživatelé, jejichž IP adresy se během připojení i několikrát mění (jak je to běžné u připojení přes modem nebo CDMA), budou mít časté potíže s přihlášením se do systému. Pokud má navíc útočník i přihlášený uživatel stejnou externí IP adresu, ani tento způsob obrany nebude mít očekávaný efekt.

Přihlášení uživatele vytvořením session je jedna z nejoblíbenějších technik. Umožňuje různé kombinování bezpečnostních principů a především dovoluje programátorům nastavit si každou drobnost chování v procesu přihlašování. K zadávání uživatelského jména a hesla se využívají HTML formuláře, které by měly bát odesílány výhradně metodou POST. Při použití metody GET se totiž informace z formulářů uloží do historie prohlížeče jako parametry adresy za otazníkem a kdokoliv si je může přečíst.

2.3.4 Metoda Challenge-Response

Je účinným vylepšením přihlašování přes sessions. Využívá skriptování na straně klienta. Výhoda této metody je, že umožňuje autentifikaci i přes nechráněné komunikační spojení bez nutnosti přenášet heslo v nekódované podobě. Systém s vygenerovanou přihlašovací stránkou zašle klientovi i náhodně vygenerovaný řetězec. Jakmile uživatel zadá své heslo, tato údaje se za využití JavaScriptu ještě u klienta zřetězí a zahešují jednocestnou hešovací funkcí, jakou je MD5. Takto získaný přihlašovací heš se odešle na server. Server samozřejmě bude znát kontrolní řetězec, který odeslal klientovi. Na své straně jej tedy spojí s heslem uživatele, které rovněž zná a po zahešování získá

stejnou hodnotu, jakou očekává i od klienta. Pokud tedy bude doručený řetězec shodný s vytvořeným kontrolním, je jisté, že uživatel zadal správné heslo a je možné jej vpustit do systému.

2.4 Všeobecné postupy při návrhu databáze

Základem úspěšné realizace systému je dobře navržená databáze. Je vhodné věnovat tomu náležitou pozornost a předem si rozmyslet a zvážit všechna úskalí, která se mohou při práci se systémem vyskytnout. Jedním z prvních kroků je správný návrh datových struktur, ve kterých budou uchována data. Někdo zastává názor, že všechny prvky, které se mohou v systému časem měnit, by měly být uloženy v databázi a k nim vytvořena obsluha, která umožní jejich změnu. Zastávám názor umístit do databáze pouze důležité prvky systému a vše, u čeho se nepředpokládá častá změna, může být uloženo v konfiguračním souboru. Do konfiguračních souborů se zapisuje snadněji, než do databáze a k nastavení aplikace není třeba programovat další nástavby v uživatelském rozhraní. Dobře strukturovaný konfigurační soubor může snadno editovat i laický uživatel. V mnoha případech se jedná o jednorázové konfigurační direktivy pro aktuální použití systému v konkrétním případě.

Při návrhu databáze by se mělo dbát na dodržení několika hlavních zásad, které jsem shrnul v následujícím seznamu:

1. Hodnoty v jednotlivých sloupcích tabulek by měly být vnitřně nedělitelné. Je lepší ukládat samostatně křestní jméno a příjmení než celé jméno do jednoho sloupce.
2. Data by měla být ukládána do sloupců správných typů. Je-li například dostupný typ DATE, je nevhodné použít pro datum typ VARCHAR.
3. Jednotlivé řádky by měly mít jednoznačný identifikátor (primární klíč) nezávislý na uložených datech. Tento identifikátor se použije při odkazování na záznam v tabulce. V MySQL lze s výhodou použít modifikátor AUTO_INCREMENT.
4. Veškerá data kromě odkazů na primární klíče by v databázi měla být uložena právě jednou. Pokud se data opakují, vede to k paměťové i výkonnostní neefektivitě a zhoršuje se možnost úpravy dat a případného rozšiřování databáze.
5. Návrh databáze by měl být pevný a webová aplikace by ani neměla mít právo modifikovat strukturu existujících tabulek nebo vytvářet tabulky nové.

2.5 Framework

Je možné vymezit skupinu funkčních požadavků, které jsou společné pro téměř všechny pokročilejší webové aplikace. Jejich podpora přímo v používaných programovacích jazycích ale často nativně neexistuje vůbec, nebo je pouze částečná či nedostatečná. Potřebné funkce si musí vývojář doprogramovat sám. Naštěstí pro naprostou většinu takových úloh existují již hotové knihovny, které bývají i volně k dispozici. Ucelenější soubor takových knihoven, zajišťující komplexní funkcionalitu

nutnou pro vývoj webové aplikace, tvoří framework. Neexistuje žádná ustálená a všeobecně přijímaná definice výrazu framework. Nejčastěji se však objevují dvě základní pojetí tohoto výrazu.

2.5.1 Vzájemně provázané knihovny

První přístup chápe framework jako balík užitečných kódů a knihoven, které uceleně a standardně pokrývají dané společné požadavky aplikací. Jsou uloženy v nějaké dostupné repository a programátor si do aplikace načítá jen ty vybrané knihovny, které zrovna potřebuje, v závislosti na charakteru dané aplikace. V komplexní aplikaci tak pro ošetření nějakého problému použije pokročilé a sofistikované knihovny, zatímco pro jednoduchou úlohu stačí knihovna se základní funkčností. Tyto knihovny jsou však komplexně provázány a samostatné použití je nesnadné či nedostatečně efektivní. Síla takového řešení spočívá v komplexním použití frameworku. Někoho napadne, že toto může být velká nevýhoda a je to skutečně pravda. Naštěstí jsou moderní frameworky komplexního typu dostatečně flexibilní, aby měl programátor nezbytnou volnost v návrhu aplikace.

2.5.2 Samostatně použitelné knihovny

Druhé pojetí je užší v tom smyslu, že vše výše popsané považuje jen za balík relativně samostatných navzájem nesouvisejících knihoven, nikoliv za framework. Framework vzniká právě až vhodným poskládáním těchto jednotlivých knihoven dohromady, jejich účelným provázáním. Vzájemnými interakcemi mezi nimi často získáme dodatečné funkčnosti, které se u izolovaných knihoven nemohou objevit. Tímto postupem se tedy vytvoří pevný rámec, framework, uvnitř kterého se pak vyvíjí výsledná aplikace.

Takovým typem frameworku je i velmi známý a populární Zend Framework (www.framework.zend.com). Jeho jádro tvoří jediná komponenta – *Front Controller*. Lze říci, že při absenci této komponenty by se tento nástroj snad ani nedal označit za framework, jelikož opravdu každá jeho podknihovna může být využita samostatně. *Front Controller* je velmi užitečnou, téměř až zásadní vlastností Zend Frameworku a v drtivé většině aplikací je využit.

2.6 Architektonické vzory

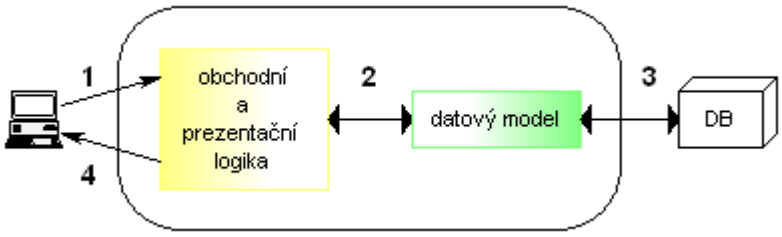
Každá aplikace je postavena na nějaké struktuře – budeme říkat architektuře. Někdy lze vidět aplikace, o nichž by bylo možné říci, že postrádají jakoukoliv architekturu, ale není tomu tak. Každá aplikace má svou architekturu, která určuje pracnost vytvoření systému, snadnost provedení upgrade, rychlost odezvy a mnoho dalších faktorů. Zvolení správné architektury je zásadní rozhodnutí. Pro vývoj webových projektů existují 2 nejznámější softwarové architektonické vzory:

1. Model-1
2. Model-2 (známější pod názvem Model-View-Controller)

V následujícím textu se architektonickými vzory budu zabývat podrobněji a pokusím se objasnit jejich výhoda a nevýhody.

2.6.1 Model-1

Je to velmi jednoduchá architektura, která kombinuje *obchodní* a *prezentační logiku* aplikace. Od uživatelského rozhraní s řídící logikou odděluje pouze datový model. Přincip komunikace v aplikaci je takový, že klient posílá požadavek přímo modulu, který provádí souhrnně zpracování požadavku, i validaci dat a generování nového pohledu zpět uživateli. Stručněji řečeno v této architektuře chybí řadič (Controller).



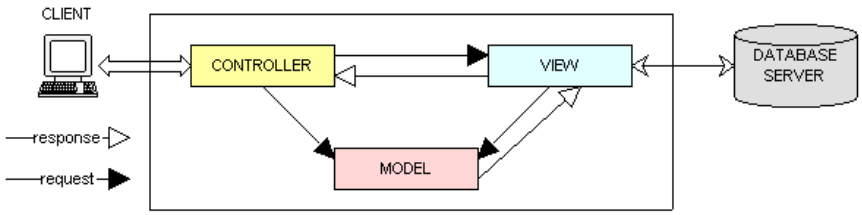
Obrázek 7 – Schéma architektury Model-1

Nevýhoda tohoto modelu je především obtížné generování pohledu uživateli a vytváření nového přístupového bodu k aplikaci. Tato architektura se nehodí pro složitější projekty, jelikož nutí programátora k opakovanému zápisu stejného kódu byť pro mírně upravené kontrolní parametry.

2.6.2 Model-View-Controller (Model-2)

Zásadní změnou oproti Modelu-1 je zavedeni komponenty *Controller*, která přejímá všechny akce od klienta a po vyhodnocení zachycené akce volá komponentu *View*. Komponenta View je dále napojena na *Model* a ten na datový sklad. View tedy komunikuje s Modelem a poskytuje klientovi zobrazení požadovaných informací opět skrze Controller.

Model-2 tedy odděluje obchodní logiku od prezentační a každá komponenta zpracovává jí náležící akce, aby byla zachována hierarchie komponent v architektuře.



Obrázek 8 – Schéma architektury Model-View-Controller

Tento architektonický vzor je v současné době nejpopulárnějším vzorem pro vytváření složitějších webových aplikací. Poskytuje struktury pro potřebné zapouzdření objektů a aplikace je

snadno rozšiřitelná. Eliminuje vznik rizika, že při náročnějším upgrade jedné systémové části bude nutné upravovat i všechny ostatní části aplikace.

V Controlleru lze s výhodou implementovat všechny systémové a globální akce, jako ověření přihlášeného uživatele či kontrolu přístupových práv. Controller bývá většinou realizován jako třída, která je vytvořena podle návrhového vzoru *Singleton* (umožňuje vytvořit pouze jednu instanci objektu a poskytuje statickou metodu pro získání této instance).

3 Administrační framework Liquid Admin

Myšlenka na vytvoření univerzálního administračního projektu se v mé hlavě objevila poprvé v roce 2006. Zabýval jsem se řešením různých administračních systémů. Vesměs se jednalo o velmi jednoduché systémy, které sloužily k ovládání webových projektů, webových katalogů, diskuzních fór a využívaly v průměru 10 databázových tabulek s jednoduchými až středně složitými vazbami.

Při vytváření takových systémů si člověk snadno povšimne, že všechny mají něco společného. Mají obdobné požadavky na bezpečnost, upgradovatelnost, přehlednost i uživatelskou přítulnost. Obdobným způsobem rovněž pracují s databázovými dotazy a zobrazují vybraná data na obrazovce webového prohlížeče. Při shrnutí těchto požadavků bude výsledkem docela přesná podoba toho, co by měl administrační framework umožňovat.

Cílem této kapitoly je popsat principy, které byly použity při návrhu administračního frameworku Liquid Admin. Na internetu jsou volně dostupné open source projekty, které v sobě zahrnují i administrační systémy. Jedná se o redakční nebo publikační systémy (Joomla, Mambo) a blogovací systémy (Textpattern, Wordpress). Jsou postaveny na různých architekturách a zahrnují v sobě jak prezentační vrstvu webové aplikace (front-end), tak administrační vrstvu (back-end). Tyto nástroje umožňují pohodlně vytvořit kompletní, ale pouze určité druhy webových systémů. Bylo by obtížné používat je k tvorbě firemních katalogů, registračních nebo rezervačních systémů.

Názor, z něhož tento projekt vychází, je takový, že prezentační část webového systému by se neměla kombinovat s administrační. Prezentace vyžaduje zcela jiný přístup než administrace. Návštěvníci webového systému mají odlišné představy o tom, jak by se měly zobrazovat informace a jaké možnosti by měl systém poskytovat. To může být pro administrátora irelevantní – důležité je rychlá a nenáročná správa databáze, souborů systému, oddělení uživatelských rolí a jiné. Administrační framework Liquid Admin bude umožňovat tvorbu právě té druhé části webového projektu – administrace. Nechovám naději v tom, že by tento systém byl někdy použit pro tvorbu bankovního klientského systému. V takových projektech se aplikují zcela jiné principy a priority. Cílem je umožnit programátorovi vytvořit administrační prostředí ke středně velkým až velmi velkým systémům v podstatně menším čase, než by tomu bylo při použití komplexních webových nástrojů a frameworků.

3.1 Požadavky frameworku

3.1.1 Funkční

Modularita - základním funkčním požadavkem je modularita frameworku. Rozdělením aplikace do logických funkčních celků bude dosaženo přehlednosti. Moduly budou snáze a rychleji upravitelné a při oddělení proměnných prostředí bude mít změna modulu z programového hlediska minimální vliv na jiný modul.

Propojitelnost modulů – každý modul bude napojen pouze na jednu databázovou tabulku, respektive žádný jiný modul nebude do stejné tabulky vkládat nová data, ani editovat existující data. Tabulky mohou mít mezi sebou vztahy 1:N i M:N a je nutné, aby bylo možné moduly snadno propojit a zajistit tak přístup i k připojeným tabulkám.

Rozšiřitelnost – předpokládá se, že moduly budou často upravovány a měly by mít tedy strukturu, která v tomto ohledu nebude vývojáře omezovat. Mělo by být snadné vytvářet rozhraní pro nové funkce modulu i pro změnu propojení stávajících funckcí.

Univerzálnost – jak již bylo zmíněno, je nutné pamatovat na to, aby framework pokrýval co možná nejširší použití a neomezoval pouze na implementaci některých funkcí. Preference a nastavení administračních systémů se mohou lišit, ale framework by neměl vývojáře omezovat.

Konfigurovatelnost – ačkoliv jádro frameworku a struktura modulů bude pevně daná, je nutné zajistit možnost konfigurace výsledného systému. Knihovny systému, které není vhodné z bezpečnostních důvodů implicitně konfigurovat, by měly být nastaveny pomocí interních konstant, jejichž modifikací lze zajistit jiné chování knihovny.

Ochrana sessions – mnoho administračních systémů je tvořeno k běžným prezentačním projektům, které bývají umístěny na sdílených serverech veřejných webhostingů. Toto řešení poskytuje novou oblast možných útoků na systém, častěji však na session data. Při správě sessions by na to mělo být pamatováno a framework by měl obsahovat knihovny, které eliminují nebo znesnadňují tento typ útoků.

Autorizace a autentifikace – s předchozím bodem souvisí přihlašování uživatelů, uchovávání jejich dat a řízení přístupu v administračním systému.

Samostatná databázová vrstva – u současných systémů bývá vhodné používat pro přístup k datovému skladu knihovny v oddělené vrstvě. Lepší řešení implementují v databázových třídách návrhový vzor a*dapter*, který umožňuje využívat stejné rozhraní pro přístup k různým databázím. Zvyšuje se tím přenositelnost aplikace tak, že není pevně závislá na zvolené databázi.

Monitorování aplikace – ve složitějších systémech s mnoha uživateli se může snadno vyskytnout chybné chování. Včasným odhalením takové chyby lze předejít komplikacím v budoucím provozu. Monitorování systému a logování akcí uživatelů pomáhá dohledat příčiny problémů.

Oddělení PHP a XHTML kódu – možnost spojování XHTML a PHP kódu je pro mnoho programátorů výhodou. Z mého hlediska jsou ale programy zapsané takovým stylem nepřehledné a programátora to odvádí od původního záměru (psát algoritmus) k tvorbě layoutu systému. Layout by měl být uložen jako šablona, nejlépe při použití jednoduchého šablonovacího systému. Implicitním šablonovacím systémem, který bude využívat framework Liquid Admin, je **TemplatePower** (http://templatepower.codocad.com).

Validace vstupních parametrů – principy a účel validace vstupních parametrů byl popsán dříve v kapitole **Bezpečnost a zebezpečení aplikace**.

3.1.2 Datové

S datovými požadavky úzce souvisí používání správných datových typů v aplikaci i databázi. Jak je známo, jazyk PHP není striktně typový. Určení typů proměnných však vnáší do programování řád, který má za účel snížení chybovosti programů a redukci neočekávaného chování. Podporu typovosti pro kód psaný v PHP je nutné přidat explicitně, nejlépe vytvořením třídy, která bude přiřazovat hodnoty proměnným a kontrolovat správnost předaných typů hodnot.

Jelikož systém bude uchovávat většinu důležitých dat v databázi, jsou datové požadavky orientovány především na správný návrh databázových tabulek. Při dodržení několika málo pravidel bude mít vývojář hotovou tabulku, kterou může přímo napojit na připravené třídy frameworku bez jejich dodatečné konfigurace. Tyto pravidla vycházejí z kapitoly **Všeobecné postupy při návrhu databáze** a rozšiřují ji o pravidla specifická pro framework Liquid Admin.

Identifikátor záznamu – každá tabulka by měla obsahovat jednoznačný identifikátor záznamu – zde se předpokládá konstrukce: **id | int(10) unsigned | not null | auto_increment | primary key**

Boolean atributy – s ohledem na kompatabilitu s předchozími verzemi databáze MySQL je nutné tento typ reprezentovat vždy jako **tinyint(1) | unsigned | default '0/1'**

Vyžadované a nevyžadované atributy – globální pravidlo, které by mělo platit u libovolného typu atributu je, že vyžadovaný atribut bude uložen vždy s nastavením **not null** a nevyžadovaný jako **null**. Implicitní hodnota není pro funkci frameworku podstatná.

Autentifikační třída, která bude v každém administračním systému vyžaduje při implicitním nastavení trochu složitější pravidla pro své dvě základní tabulky.

Tabulka *accounts*, která bude obsahovat informace o všech účtech, musí obsahuvat minimálně atributy: **id, login, pass, name, perms, active**.

Tabulka *ac_sessions* bude obsahovat informace o vytvořené session (relace přihlášeného uživatele) s atributy: **id, id_account, session, ident, last_action**.

Podrobněji bude struktura tabulek popsána v příloze této práce.

3.2 Architektura aplikace

Architektura je základem celého systému. V předchozích kapitolách bylo zmíněno mnoho o populární architektuře Model-View-Controller (MVC), avšak tato architektura má svá onezení. Velmi vhodné je použití při tvorbě systému v plně objektových jazycích, jakým je Java. Dokonce i platforma .NET nabádá k používání této architektury, přičemž existuje celá řada komponent přímo určených k implementaci do MVC. Cílem této práce je rovněž vytvoření webového systému, avšak zcela odlišného od běžných aplikací. Administrace má jiné priority než webové portály. Pro tvorbu administračního systému již použití architektury MVC není tak revolučním nápadem.

Většina administračních systémů má jeden společný rys – všechny modely mají velmi podobnou prezentační vrstvu. Z tohoto pohledu je vytváření samostatné vrstvy View ke každému modelu zbytečné a spíše to programátora zatěžuje. Logika takového systému je oddělením vrstvy View spíše složitější. Nejedná se o žádnou radikální újmu na programové přehlednosti, ale stejně tak není chybou sjednotit, případně vytvořit univerzální vrstvu View. Co je v administraci podstatné, je navigace v aplikaci a modulech, správná kontrola předávaných parametrů a bezkonfliktní chování aplikace v multiuživatelském prostředí. Tyto důvody vedly k vytvoření administračního frameworku, který využívá architekturu Model-1 a dovoluje programátorovi vytvoření mnohem kompaktnějších modulů, než by tomu bylo v případě MVC.

3.2.1 Návrhové vzory tříd

Balíček tříd, které jsou v rámci frameworku k dispozici tvoří asi polovinu objemu projektu. Pokládám tedy za důležité zmínit strukturu, jaká byla pro jejich návrh použita a popsat hlavní výhody, které vybrané řešení obsahuje.

Návrhový vzor představuje obecné řešení problému, který se opakovaně objevuje při návrhu softwaru. Návrhový vzor není knihovnou nebo částí zdrojového kódu, která by se dala přímo vložit do programu. Jedná se o popis či šablonu, jak řešit problém způsobem, který může být použit v různých situacích. Objektově orientované návrhové vzory typicky ukazují vztahy a interakce mezi třídami a objekty, aniž by určovaly implementaci konkrétní třídy. Algoritmy nejsou považovány za návrhové vzory, protože řeší výpočetní problémy a nikoliv návrhové.

Adapter – samotný název napovídá, že se jedná o návrhový vzor, který má za účel přizpůsobit rozhraní jedné třídy na rozhraní jiné. Velmi užitečné je použít tento vzor v případě, kdy potřebujeme zajistit zpětnou kompatibilitu třídy bez toho, abychom ji modifikovali. Adapter je třída, která zajišťuje funkcionalitu systému, ale neimplementuje požadované rozhraní. Existují dvě známá řešení tohoto návrhového vzoru. První spočívá v odvození nové třídy od staré nevyhovující a přidání metod, které zajistí implementaci požadovaného rozhraní.

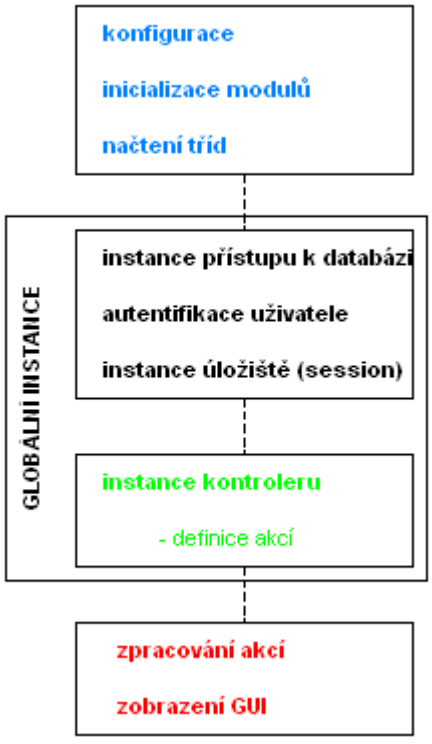
Druhý přístup využívá vložení reference na starou třídu a přeposílání požadavků na metody této staré třídy. V takovém případě může jeden adapter zajišťovat komunikaci s více objekty.

Singleton – účelem vzoru singleton je zajistit existenci pouze jedné instance takové třídy. Tato třída musí mít definovánu statickou metodu getInstance(), která vrátí instanci vytvořené třídy. Tím je zajištěna ochrana proti vytvoření více instancí voláním new Object(). Vzor singleton nachází své využití u tříd, které mají na starosti globální akce, jako je autentifikace uživatele, kontrola oprávnění po vstupu do aplikace či přístup k databázi (resp. vytvoření instance třídy pro připojení k databázi).

Factory method – v programu je někdy třeba, aby byl schopen až na základě vstupních parametrů vytvořit instanci některé ze sady tříd a tuto třídu pak používat. S výhodou se toto řešení používá při výběru třídy pro připojení k různým databázím. Pro každou databázi je vytvořena jedna třída, která implementuje požadované rozhraní a existuje třída, která obsahuje factory method. Factory method vybírá, která třída bude zvolena pro vytvoření nové instance. Třída vytvářejícího objektu je definována jako abstraktní nebo jako konkrétní (přímo implementující factory method).

3.2.2 Struktura modulu

Všechny moduly systému jsou založeny na stejné struktuře, která definuje vždy společné části i unikátní zpracování požadavků, které vysílá uživatel a dále distribuuje kontroler systému. Blíže popisuje strukturu modulu následující diagram.



Obrázek 9 – Struktura modulů frameworku

Na začátku každého modulu je třeba načíst potřebné konfigurace a provést inicializace modulů. Aktuální modul může být napojen na libovolný počet dalších modulů (vyhledávací modul, modul zobrazující závislou entitu, …) a tedy je třeba, aby věděl, jak se ostatní moduly jmenují a kde je má nalézt. Následuje načtení potřebných tříd, které modul využívá. Zde se načítají třídy pro přístup k databázi, autentifikační třída a rozhraní kontroleru a libovolné další uživatelsky definované třídy. Pro automatizované načítání tříd lze v jazyku PHP s výhodou použít metodu __autoload(), která automaticky vyhledá php soubor s obsahem potřebné třídy a vloží jej do prováděného skriptu. Je však nutné, aby se všechny soubory jmenovaly podle zavedeného vzoru. Nechceme-li uživatele svazovat v pojmenovávání souborů tříd, pak je lepší zavádět třídy klasickým způsobem pomocí require_once().

V každém modulu je nutno vytvořit instanci třídy, která se stará o provádění databázových dotazů. Ve frameworku Liquid Admin lze použít vytvořenou třídu *DB*, která je vytvořena podle návrhového vzoru singleton a implementuje mimo jiné i zpětné procházení výsledkem dotazu (více informací o této třídě je uvedeno v příloze). Do stejného bloku je zařezeno rovněž vytvoření instancí tříd pro autentifikaci uživatele a definici skladu persistentních parametrů, bez něhož framework nelze efektivně používat. Implicitně je jako datový sklad určen soubor se session přihlášeného uživatele. Bezpečnostní rizika sessions a opatření proti zcizení identity jsou dále uvedeny v kapitole 3.2.4 Bezpečnostní prvky.

Nejdůležitější částí každého modulu je však kontroler. Kontroler přebírá parametry, které zaslal uživatel a po jejich zpracování volá příslušné akce. Administrační systém je mnohdy velmi citlivý na uživatelská oprávnění a řízení systému je nutné tomu přizpůsobit. Kontroler v tomto případě přebírá i nastavení uživatelských práv a po vyhodnocení může být určená část modulu kompletně uzamčena.

Červenou barvou je v grafu vyznačena část struktury modulu, kde jsou implementovány metody, které zpracují požadované akce. Tvoří nejpodstatnější a největší část každého modulu. Tyto metody využívají často i jiné definované třídy, které pomáhají zpracovávat automatizované úlohy. Bez implementace objektového přístupu k aplikaci by moduly frameworku mohly být zbytečně několikrát větší. Automatizace často prováděných činností zvyšuje úsporu kódu i přehlednost celého řešení.

3.2.3 Integrace modulu

Každý modul systému má právo zapisovat a editovat data pouze jedné databázové tabulky. Tak je framework postaven a ačkoliv to je možné, bylo by chybou toto pravidlo nerespektovat. Tento princip dovoluje separovat složité řízení databázových vztahů do jednoduchých kroků. Řešení vyžaduje definici vztahů mezi moduly a zavedení (integrování) modulů s definovanými vztahy do kostry systému.

Modul je integrován do systému svým zařazením do vícerozměrného pole $MODULES, které má následující strukturu:

```
$MODULES['module.php'] = array(
    'menu' => 'Název modulu',
    'title' => 'Titulek modulu',
    'perms' => 'A',
    'submodules' => array('mododule2.php')
);
```

Indexace prvního rozměru pole probíhá podle názvu souboru modulu. Druhý rozměr pole obsahuje klíče menu, title, perms, submodules. Tyto klíče mají specifické vlastnosti a lze jimi nastavit chování i zobrazovanou hodnotu modulu.

menu – obsahuje název modulu, který je zobrazen v menu administračního systému

title – titulek modulu je zobrazen na začátku výstupu grafického uživatelského rozhraní (může obsahovat delší popis, zatímco v menu by měla být maximálně dvouslovná definice)

perms – uživatelská práva, která modul vyžaduje (framework obsahuje jednoduché řízení práv, které je možné libovolně rozšířit – toto bude blíže vysvětleno později)

submodules – při modelování vztahů 1:N a složitějších je někdy nutné, aby podmodul, který řídí zobrazení závislé entity nebylo možné zobrazit jinak, než s vazbou na konkrétní rodičovský záznam. K tomu slouží tato volba. Existuje-li k modulu nějaký submodul, bude uveden zde. Jelikož hodnotou je opět pole, může jeden modul mít více podmodulů.

3.2.4 Bezpečnostní prvky

Informační systémy uchovávají většinu svých dat v databázích. Toto řešení přináší výhody v rychlosti ukládání a vyhledávání dat a také nové bezpečnostní prvky oproti datům, která jsou uložena v nechráněných souborech. Toto oddělení aplikační a datové vrstvy přináší ovšem i mnohá nová potencionální ohrožení dat. Ukládání dat do databází sice eliminuje některé zásadní bespečnostní nedostatky, ale rovněž vytváří prostor pro zcela nový druh útoků, jako *SQL injection* a související útoky, k nimž patří i méně známý *privilege escalation* (http://en.wikipedia.org/wiki/Cross-calation).

Administrační systémy oproti obecným informačním systémům ještě navíc zahrnují téměř vždy rozdělené role přístupu k editaci informací. Velmi oblíbeným zabezpečovacím prvkem je autentifikace, většinou řešená prostřednictví *sessions*. Systém Liquid Admin tuto techniku využívá rovněž. Technika přihlašování pomocí sessions s sebou přináší nutnost správné implementace, protože riziko napadnutí session přichází jak ze strany webového serveru, tak i z bezpečnostních děr v některých verzích interpretu jazyka PHP. Asi nejznámějším pojmem z oblasti bezpečnosti sessions je *session hijacking.*

Session hijacking – obecný pojem pro získání přístupu k názvu cizí session nebo datům, která tato session obsahuje. Zahrnuje techniky, které umožňují napadení session a získání těchto citlivých

informací ve prospěch útočníka. Cílem použití této techniky je však vždy získání přístupu do neoprávněné oblasti, kam zpravidla má přístup oběť útoku. Techniky pro útok jsou **session fixation**, **sidejacking**, **session file stealing**, **cross-site scripting**.

1. **Session fixation**

   Cílem je přednastavit identifikátor session oběti při přihlášení do aplikace tak, aby jej útočník znal a mohl použít pro záměnu své identity a vadávat se tak za oběť.

   Scénář útoku:

   - Jana se přihlašuje do systému www.unsafe.cz, který akceptuje přednastavený session identifikátor
   - Petr pošle Janě e-mail, v němž uvede návodný text, aby se Jana přihlásila do svého účtu na adrese www.unsafe.cz/?SID=PETER_KNOW_IDENTIFIER
   - Jana se přihlásí do systému na uvedené adrese.
   - Petr se musí dozvědět, že se Jana přihlásila do systému a poté použije známý identifikátor k získání přístupu do stejné části systému, jako Jana.

   Obrana proti útoku:

   - Znemožnění nastavit session identifikátor prostřednictvím GET a POST superglobálních proměnných – používat pouze cookies (obtížnější nastavení, avšak ne neproveditelné – často však od útoku odradí, protože je nutná větší interakce útočníka a oběti).
   - Využívat SSL / TSL session identifikátory – toto řešení však mnohé hostingové služby neumožňují – vhodné pro robustní aplikace internetového bankovnictví.
   - Regenerace SID při každém požadavku na server nebo alespoň při prvním přihlášení – takto bude útočníkovo předem nastavené SID nepoužitelné. Toto je spolehlivé řešení.
   - Kontrola serverových proměnných HTTP_REFERER, USER_AGENT a IP adresy – tedy při přihlášení oběti se v systému může uchovat například hash z použitého prohlížeče a IP adresy. Útočník by musel mít stejnou IP adresu a používat stejný prohlížeč, jako útočník. V opačném případě systém pro daný SID identifikátor nepovolí přihlášení, případně je možné, aby dočasně přihlášení z útočníkovy IP adresy zablokoval, což velmi znesnadní obcházení této techniky ošetření.

   Aplikované principy v systému Liquid Admin:

   - Uchování identifikátoru SID je možné pouze v cookies.
   - Při každém přihlášení je SID regenreován.
   - Při přihlášení je možné zvolit si, zda uživatel chce pokročilejší bezpečnostní opatření. Toto spočívá v kontrole používaného prohlížeče a IP adresy. Volba je ponechána na uživateli, jelikož je možné, že bude mít proměnlivou IP adresu během připojení.

2. **Sidejacking**

   Tento typ útoku na session vyžaduje, aby utočník i oběť byli oba v jedné síti. Útočník používá sniffing paketů, tedy čtení paketů, které putují sítí. Takto může číst komunikaci oběti s cílovým serverem, přičemž není problém vyfiltrovat z komunikace obsahy cookies, v nichž jsou data session identifikátoru. Poté, co se útočník dozví SID, probíhá napadení stejně, jako v předchozím bodě. Nezabezpečené Wi-Fi sítě jsou tímto typem útoku napadnutelné zvláště, pokud existují jako hot-spoty, k nimž se může připojit kdokoliv a složení připojených uživatelů je naprosto neevidované.

   Obrana proti útoku:

   - Šifrování komunikace pomocí SSL – toto je jeden z nejbezpečnějších a nejúčinnějších prvků pro zabránění odposlechu komunikace v síti. Odposlech šifrované komunikace je natolik složitý a časově náročný, že pro útok na systémy, kde session trvá v řádu desítek minut až hodin je reálně neproveditelný.

3. **Session file stealing**

   Má-li útočník fyzický přístup k disku, na němž se uchovávají data session oběti, pak je útok asi nejjednodušší. Odcizení celého souboru se session dává útočníkovi jak znalost identifikátoru, tak i veškerých dat v session.

   Obrana proti útoku:

   - Dostatečná ochrana fyzických částí serveru a spolehlivá obsluha serveru. V nejlepším případě by měla být monitorována veškerá aktivita na serveru, aby bylo možné dohledat odcizení session.

4. **Cross-site scripting**

   Jinak často označován jako XSS je využíván k získání přístupu do aplikace nebo k vyvolání škodlivého kódu. Útok je mnohdy spojován se session fixation útokem, jelikož poskytuje nové techniky, jak odhalit session identifikátor podstrčením zákeřného kódu do webové aplikace. Rozlišujeme různé druhy tohoto útoku - DOM-based (Type 0), Non-Persistent (Type 1) a Persistent (Type 2). Nadále bude popsán pouze typ 2, jelikož je to v souvislosti s odcizením session nejčastěji používaná metoda.

   Scénář útoku:

   - Bobův web umožňuje členům vkládat příspěvky a zároveŇ je tento web náchylný k XSS Type 2 útoku.
   - Petr vytvoří atraktivní příspěvek, který obsahuje škodlivý kód a vloží jej do Bobova webového systému. Poté naláká ostatní uživatele, aby si jej prohlédli.
   - Webový příspěvek obsahoval JavaScriptový kód, který nyní každému prohlížejícímu uživateli prohledá cookies a odešle session identifikátory nebo jiné citlivé informace Petrovi (například pomocí Ajaxové technologie).
   - Petr nyní může využívat získané údaje k přihlášení se pod zcela odlišnými identitami.

Obrana proti útoku:

- Jediná účinná obrana proti tomuto typu útoku je kontrola vstupů. Je nutné znát očekávaná data, která budou uživatelé do systému vkládat a kontrolovat existenci škodlivých dat. Je účinné odstraňovat všechny HTML tagy nebo povolovat pouze některé. Dále je důležité vhodně escapovat vstupní data. To jako druhotný účinek zabrání SQL injection.

Aplikované principy v systému Liquid Admin:

- V systému není umožňeno provádění jakéhokoliv uživatelsky definovaného kódu. Veškerý HTML kód je zobrazen s nahrazenými HTML značkami za jejich ekvivalentní entity.

V souvislosti s bezpečnostními technikami existuje ještě jedna hrozba, která nespadá do kategorie útoků session hijacking. Jedná se o **session poisoning**. Název samotný napovídá, že se jedná o „otrávení“, respektive nakažení session proměnných podvodnými daty, které mohou útočníkovi umožnit vydávat se za jiného uživatele nebo vlastnit jiná práva přístupu.

Představte si situaci, kdy by v session byl uchováván příznak přihlášeného uživatele, jeho oprávnění nebo dokonce kontrolní číslo účtu. Přenastavením session by pak mohl uživatel dostat úplně jinou roli přístupu nebo ovládat naprosto odlišný účet.

Session poisoning se projevuje v systémech, které běží na sdílených serverech. Sdílený server je takový, na němž je prostor pro aplikace rozdělen mezi více uživatelů. Na mnoha takových serverech bývá zvykem ukládat session soubory do jednoho určeného společného adresáře. Takový adresář samozřejmě nebývá zpřístupněn pro čtení ani zápis žádnému uživateli. Je to systémový adresář a proto pouze systém do něj může přistupovat. Zdá se, že napadnout tento adresář i soubory v něm je tak nemožné. To však není tak zcela pravdou. To, že do adresáře přistupuje systém samotný se pozná podle jeho oprávnění, protože i systém má své zóny přístupu odděleny oprávněním. Útok tedy spočívá v oklamání práv systému. Proveení je překvapivě jednoduché, pokud si uvědomíme základní souvislost. Skript, který na server nahraje uživatel má sice oprávnění daného uživatele, ale není problém vytvořit skript, který vygeneruje jiný skript (obsahující scénář útoku). Takový nově vygenerovaný skript byl již vytvořen systémem a proto má i systémová oprávnění a může tedy bez problému přistupovat do systémových složek.

Obrana proti tomuto útoku je ovšem možná. Stačí, aby pro každý účet na serveru byly session proměnné uchovávány v místě každého uživatele zvlášť. Tedy dva rozdílní uživatelé serveru budou mít dvě různá místa pro uložení session souborů a oprávnění pro vstup do těchto adresářů bude nastaveno na příslušného uživatele.

Naneštěstí výše popsaná metoda není jediným typem útoku, který můžeme označit za session poisoning. Spatný návrh aplikace může umožnit změnu obsahu session pomocí injekcí v URL. Vezměme v úvahu následující evidentně špatný kód:

```
$var = $_GET["something"];
$_SESSION["$var"] = $var2;
```

Lze vidět, že session proměnná je nastavována podle parametru GET, pradaném v URL. Pokud by session obsahovala důležitou proměnnou $_SESSION["permissions"], která je kontrolována pro nastavení oprávnění, mohli bychom systém spustit s přidanými prapetry v URL takto:

```
vulnerable.php?permissions=superadmin
```

Uživatel by tak dostal nejvyšší oprávnění a mohl se systémem provádět libovolné administrátorské operace. Je evidentní, že takto je v systému velké bezpečnostní rizio.

Obranou proti takovým útokům je validace vstupních parametrů a správné nastavení serveru. Velkou chybou bývá nastavení direktivy v souboru php.ini: register_globals = on. Superglobální proměnné mohou v systému vyvolávat neočekávané chování a velmi špatně se kontrolují.

Framework Liquid Admin je navržen tak, aby do session nezapisoval citlivé informace, tedy oprávnění a kontrola uživatele se při každém požadavku do systému kontroluje znovu a vychází z databázového nastavení, které je porovnáváno pouze na aktuální session identifikátor. Proměnná $_SESSION obsahuje pouze nastavení modulů, jako stránkování, filtrace údajů a obdobné vedlejší funkce. Je samozřejmě nepříjemné a nežádoucí, aby někdo modifikoval i tato data, ale rozhodně to nemůže vést k vážnějším bezpečnostním dírám, které by umožnily získat kontrolu nad systémem.

3.3 Implementované principy

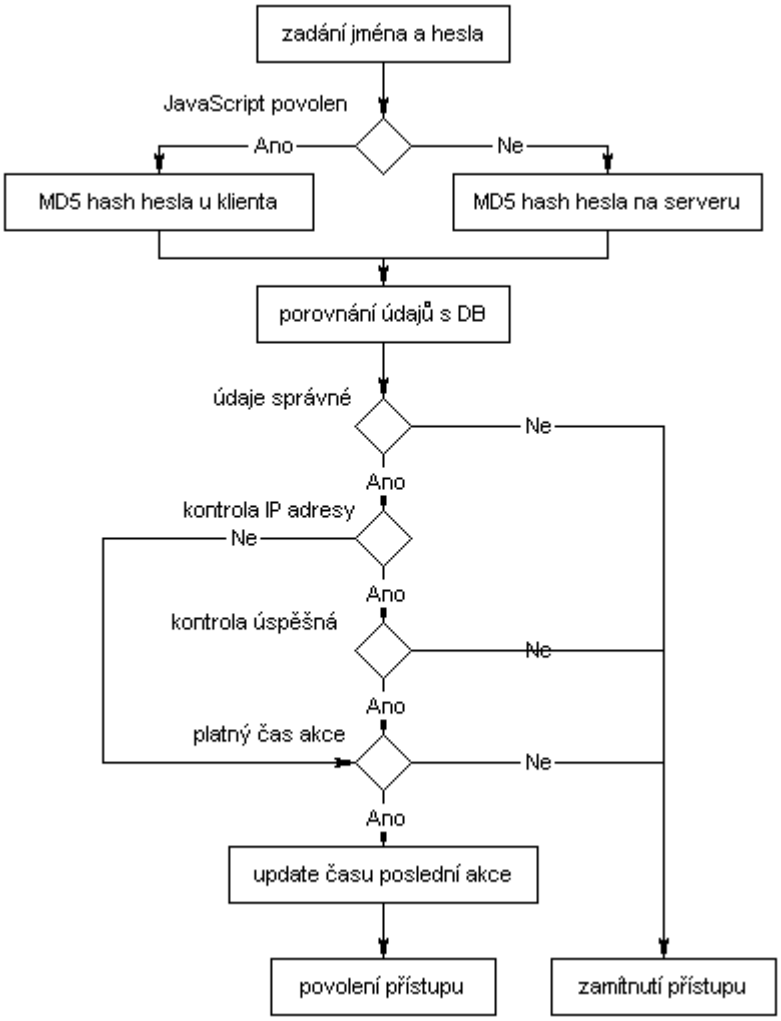
3.3.1 Autentifikace uživatele

Základní úkon, jaký musí uživatel při přihlášení do administrace provést, je přihlášení. Na tuto část systému je kladen velký důraz. Systém může být sebe lépe zpracován, ale když není dostatečně zabezpečen, nelze jej používat v reálném nasazení.

Pro účely autentifikace jsou ve frameworku implementovány ochranné prvky jak v klientské, tak v serverové části. V klientské části jsou dostupné soubory JavaScriptových funkcí, zahrnující funkce na hashování citlivých údajů, které by jinak putovaly sítí v nechráněné formě. Přihlašování by však mělo být vždy navrženo tak, aby neomezovalo uživatele, kteří nepoužívají JavaScript.

Serverová část má za úkol prověřit, zda přihlašující se uživatel má povoleno ukládat cookies, jelikož cookies bude jediná možnost. jak uchovat session identifikátor (SID). To lze provést prostým principem, kdy na přihlašovací straně nastavíme testovací cookie a po odeslání údajů zkontrolujeme, zda testovací cookie dorazila rovněž na server.

Následuje skutečné ověření přihlašujícího se uživatele oproti záznamům v databázi uživatelů. Tuto funkci zajišťuje autentifikační třída *Auth()*, která dále obsahuje metody pro určení času nečinnosti přihlášeného uživatele a získání hashe z informací o použitém prohlížeči a IP adrese uživatele. Přihlašování je možné nastavit požadovaným způsobem, který blíže popisuje následující diagram.



Obrázek 10 – Schéma průběhu autentifikace

Autentifikační třída implementuje veškeré zásadní techniky, které by měly ochránit jakýkoliv administrační systém založený na frameworku Liquid Admin proti průniku cizích osob. Jedná se o prosté techniky, jako regenerace SID při přihlášení a kontrolu IP adresy uživatele. Tuto třídu je možné nastavit libovolným způsobem, případně může být použita jako předek jiné třídy, která od ní bude odvozena a bude implementovat další požadované metody, jako autentifikace pomocí SSL.

3.3.2 Autorizace uživatele

Autorizace označuje získání přístupu k informacím, funkcím a dalším objektům, který se skládá z:

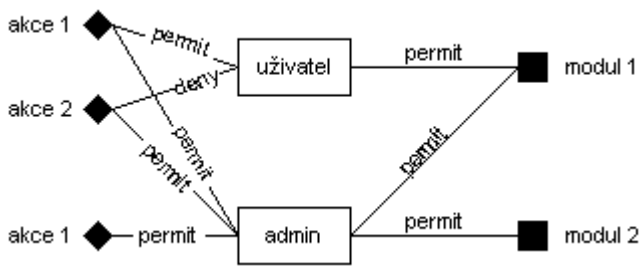
- autentizace subjektu (zjištění jeho identity – viz. třída *Auth()*)
- vyhledání v seznamu oprávněných subjektů, jejich rolí a práv
- udělení oprávnění nebo odepření přístupu

Seznam oprávnění je v informačních systémech realizován přidělením oprávnění na soubory, adresáře, moduly, provedení operace nebo přístupu k jiným prostředkům v systému. Autorizaci provádí zpravidla vyhrazená třída frameworku na základě seznamů pro řízení přístupu.

Nejčastější způsob autorizace uživatele je řízení přístupu založené na rolích. Role přiděluje administrátor aplikace. Ten také musí rozlišit potřeby a definovat práva všem rolím s ohledem na náročnost zdroje informací. Současně je dobré řídit se pravidlem „přidělení nejmenšího nutného oprávnění“. V důsledku tedy uživatel dostane pouze taková práva, která jsou nutná k jeho práci v systému, nikoliv už ta, která jsou možná pro nějakou budoucí akci.

Implicitní řešení autorizace probíhá ve třídě *ActionController()* frameworku. Základní ideou při řešení implementace přístupových práv frameworku byla jednoduchost. Zadávání a změna práv musí být jednoduchá a přitom dostatečně účinná, aby pokryla potřeby široké škály administračních systémů. Aby bylo možné systém takového řízení práv navrhnout, je třeba zvážit oblasti, které musí uživatel ovládat. Vzhledem k koncepci návrhu, kde každý modul obsluhuje jednu databázovou tabulku, pak by mělo být především možné povolit či zamezit používání kompletního modulu. Všechny činnosti modulu řídí kontroller, který by měl ovládat také přístup uživatelů k akcím modulu.

Výhodnou volbou pro implementaci tohoto způsobu řízení přístupu jsou *access lists*, neboli přístupové seznamy. Princip činnosti je takový, že se nejdříve do systému zavedou role. Každá role může obsahovat více uživatelů. Všichni uživatelé dané role budou mít pak práva, která roli přísluší. Většinou bývá nezvyklé, aby každý uživatel měl naprosto rozdílné určení práv, než všichni ostatní. Role tedy představují jakési skupiny, kam je možné uživatele zařadit a tím mu určit povolené akce. Druhým vstupem systému musí být seznam zdrojů, neboli akcí. Je důležité, aby systém obsahoval kompletní seznam zdrojů, které mají mít určena práva. Pokud nebude nějaký zdroj v systému definován, nebude jej možné používat a navíc jej nebude možné ani monitorovat (systém o tomto zdroji nebude vůbec vědět). Následující graf ilustruje propojení rolí a zdrojů v systému:



V předchozí ilustraci je vidět, že roli je možné propojit s modulem a tím povolit přístup ke všem akcím. Aby systém akce znal, musí být někde definován jejich seznam. Je-li v modulu některá akce přístupná pouze některé roli, je nezbytné mít možnost jiné roli přístup explicitně zakázat. Rozdělení zdrojů tedy můžeme chápat jako kompletní moduly nebo seznamy akcí. Je třeba nezapomínat, že různé moduly mohou mít pojmenovány některé akce stejně a uživatel by neměl být složitě omezován při výběru názvů akcí. Popsaný způsob je již dle obrázku značně složitý a pro většinu případů je třeba definovat přístup modulu jako celku. Kontrola práv je tedy implicitně sjednocena s kontrolerem a funguje tak, že definuje přístup role k modulu.

Samotnou podstatou access listů je to, že úzce spolupracují s kontrolerem, který si vždy vyžádá ověření přístupu ke zdroji a podle obdržené odpovědi provede nasměrování uživatele na obslužnou část systému.

Zůstává nezodpovězena otázka, kam definici access listů umístit – do každého modulu samostatně nebo raději blízko hlavního kontroleru? Je nutné uvědomit si, že hlavní věcí by měla být možnost snadno upravovat role a nastavení zdrojů. Z tohoto pohledu je dobré centralizované umístění access listů u hlavního kontroleru. Má to jednu nevýhodu – při definování access listů je třeba znát všechny zdroje systému dopředu, ačkoliv při práci s aktuálním modulem tento neposkytuje žádné informace o ostatních modulech – pouze definuje podřazené moduly. Pěkné oddělení nezávislosti modulů se tím trochu pokazí na úkor snadnělšího řešení bezpečnosti. Hlavní podstata, snadná úprava access listů, zůstane nade vše pochybnosti přehledná, jelikož se bude soustředit na jednom místě. Při vhodném návrhu metod třídy je vytvoření třídy pro manipulaci access listů otázkou desítek až stovek řádků kódu. Lze to považovat za minimum při srovnání s ostatními třídami a knihovnami frameworku.

3.3.3 Navigace v modulech

Funkcionalita každého systému je do značné míry určena vhodným propojením jeho částí. Navigace v modulech systému hraje důležitou roli při už návrhu. Mimo správný chod modulu je systém vyvolávání metod, jako reakcí na uživatelské akce, přední část frameworku.

Vytvořené administrační systémy budou vždy řízeny z adresního řádku prohlížeče, odkud budou přebírat parametry pro své spuštění. Jelikož protokol HTTP je bezstavový, jediný způsob udržování kontextu je v proměnné session, v cookies nebo předáváním parametrů. Session proměnné již budou využívány ke zcela jiným druhům udržování kontextu, než je ovládání akcí. Cookies nelze použít z bezpečnostního důvodu. Nehledě na fakt, že je přinejmenším nepohodlné neustále ukládat a vyvolávat data z cookies. Parametry v URL byly vyhodnoceny jako optimální řešení. Takto předané parametry budou vždy dostupné v proměnné GET (parametry v POST budou sloužit k odesílání dat systému ke zpracování). Příklady ovládání modulu systému:

- Provedení implicitní akce modulu: http://localhost/administrace/module.php
- Editace položky s databázovým ID 5: http://localhost/administrace/module.php?action=edit&id=5
- Smazání položky s ID 5 a předání dodatečných parametrů: http://localhost/administrace/module.php?action=delete&id=5&msglabel=2

Snadno lze odpozorovat, že takto nastavený framework používá parametr **action** k vyvolání akce modulu a předáním parametru **id** se příslušná hodnota předá metodě ke zpracování. Při odesílání dat z formuláře ke zpracování se navíc ještě bere v úvahu obsah superglobální proměnné POST. Framework obsahuje množství knihoven, které je možné nastavit, aby automaticky rozpoznávaly žádosti o provedení akce a přizpůsobily vygenerování výstupu aktuálnímu kontextu aplikace.

Jednou z podstatných knihoven systému, které ušetří programátorovi velké množství práce je *MySQLDataTable*. Je to databázový adaptér, který je schopen z databázového dotazu vygenerovat tabulku záznamů, každému záznamu přidat libovolné akce a namapovat tyto akce na jedinečný identifikátor záznamu. Mimo to umožňuje provádět nad záznamy estetické i vylepšující operace. Podrobný popis této knihovny naleznete v příloze, zde je uvedena ukázka jejího použití. Přidané akce přebírá rovněž kontroler.

```
$db -> quoteInto("SELECT o.id, o.name as `Jméno`, o.surname as
`Příjmení`, o.position as `Pozice`, o.email as `E-mail` FROM
contacts o WHERE (o.id_customer = ?) ORDER BY o.surname, o.name",
$stor -> read('id'), DB::INT_TYPE);

$table = new mysqlDataTable($db->getResult(),'thin','even','odd');
$table -> commands('id', array('edit' => 'Editovat', 'delete' =>
'Smazat'));
$table -> maxLength('Poznámky', 40);
$table -> lineBreaks('real');
$table -> render();
```

3.3.4 Logování informací

Dohledávání změn v informačních systémech není snadnou záležitostí. Uchovávání dat o akcích uživatelů v systému je snad ještě komlikovanější, než uchovávání dat samotných. Většina serverů loguje přístupy uživatelů do speciálních logů, které bývají uloženy v souborech. Toto je nejzákladnější logování informací. V těchto záznamech lze dohledat datum a čas přístupu, IP adresu a celá reprezentace dotazu na server (request). Tyto záznamy bývají organizovány podle vytvořených standardů. Jedním z nich je například dokument RFC 3164 - The BSD syslog Protocol. Tyto logy obsahují záznamy o přístupu na server bez rozlišení uživatelů, což je v administračních systémech

nutností, aby bylo možné dohledat iniciátora akce. Aplikace musejí tedy vytvořit většinou vlastní systém logování údajů. Ukládání dat do databáze je pro tyto účely zřejmě nejvhodnější řešení. Pomocí SQL dotazů lze dohledat pohodlně a přesně záznamy konkrétního uživatele v definovaném čase.

Jedním z palčivých problémů je narůstající velikost databázových tabulek s logy. Tento nárůst je ovlivněn množstvím detailů, které systém potřebuje logovat. Obecně není nutné ukládat každou provedenou akci. V redakčních systémech postačí například logování informací o editaci záznamů. Správná volba oblasti logování ulehčí správu logovací tabulky.

Druhá nevýhoda logování poukazuje na rychlost aplikace, která musí logování zajišťovat. Při provedení každé akce je nutné provádět ještě další akci, která zaznamená data do logu. V některých systémech je nutné uvažovat i tuto oblast, jelikož mohou být zpracovány složitě a logování informací by mohlo zpomalit práci se systémem. Vyplatí se řešit tuto situaci spíše optimalizací logování, než úplným vynecháním. Logy poskytují cenné informace v případech, kdy dojde ke zneužití systému nebo neočekávaným chybám a připravit se o tyto znalosti není dobrým řešením.

Framework Liquid Admin poskytuje pro tento účel knihovnu pro logování informací, kterou je možné napojit na databázi. Vhodná struktura databáze je obsahuje například tyto atributy:

- **id** – identifikátor záznamu
- **id_account** – číslo účtu iniciátora akce
- **datetime** – datum a čas záznamu
- **action_name** – název akce, která je předmětem záznamu
- **action_details** – podrobnosti o akci (může obsahovat i strukturovaná data)

Příklad záznamů v logovací tabulce:

```
"19","1","2008-02-17 03:06:41","login_failed","127.0.0.1"
"20","1","2008-02-17 03:06:47","login_success","127.0.0.1"
"21","1","2008-02-17 03:13:53","logout_manual",NULL
```

3.4 Generátor modulů

Tvorba modulů pro systém, který bude založen na frameworku Liquid Admin, zabírá největší podíl práce programátora. Nabízí se úvaha, jak tento proces zrychlit. Jak usnadnit tvorbu kostry systému a hlavních částí modulů. Administraci tvoří z valné většiny formuláře a metody, které zachytávají odeslaná data. Takový systém je plný potvrzování a uživatelských vstupů.

Framework obsahuje knihovnu, která slouží k tvorbě formulářů. Výhodou používání této knihovny oproti ručnímu psaní formulářů je, že knihovna je schopna převzít validační výstup jiné knihovny a provádět nad formulářem různé užitečné akce. Patří mezi ně předvyplnění odeslaných

hodnot či zobrazení chybových textů ke každému poli.Formulář se na stránce stává přehledný a snadno editovatelný. Jak už bylo řečeno, tvorba formulářů pokrývá asi polovinu práce na celém modulu, pomineme-li jeho zařazení do aplikace a provázání s ostatními moduly. Není však nutné, aby programátor ručně psal všechny formuláře, když je systém může vygenerovat v dostatečně přesné podobě, jaká se očekává. Generátor modulů tuto činnost udělá za programátora, který pouze zkontroluje přesnost a formuláře doladí.

Princip generování modulů je postaven na faktu, že každý modul obsluhuje jednu databázovou tabulku. Můžeme-li takto modul napojit na databázi, pak není velký problém vytvořit reverzní technikou formulář, který bude vygenerován ze schématu databázové tabulky.

Vstuoním parametrem generátoru je název tabulky, která bude sloužit jako předloha. Pomocí SQL dotazu SHOW COLUMNS FROM tablename je možné získat podrobné informace o atributech tabulky. Každý atribut má definovány parametry:

Field – název atributu

Type – typ hodnoty (int, varchar, datetime, text, zda je číslo typu unsigned a rovněž povolená délka hodnoty v tomto atributu)

Null – informace, zda hodnota je vyžadována, nebo zda je povoleno vkládat NULL hodnotu

Key – rozeznává dva druhy klíčů - primary a unique (v databázích je ještě možné použít obyčejný klíč k indexaci, ale ten pro konstrukci formuláře není nikdy potřeba)

Default – hodnota, kterou bude atribut obsahovat, nebude-li explicitně definována uživatelem

Extra – obsahuje hodnotu auto_increment, je-li takto atribut definován

Na základě těchto informací jsme schopni vygenerovat dostatečně přesný formulář. Nejjednodušší je situace v případě typu hodnoty **text**. Pro zadávání této hodnoty se téměř výhradně používá párový HTML prvek <textarea>. V databázích se však mohou ukládat další typy dat. Varchar, char, integer a float bývají zobrazovány jako <input type="text" … />. Tagy input mohou obsahovat i další informace, jako implicitní hodnotu a maximální povolenou délku. Tyto informace není těžké získat z výše uvedených parametrů databázového atributu.

Pro efektivní využití generátoru je však nutno dodržovat i některá pravidla. Jedním z takových pravidel je například to, že boolean hodnoty by se do databáze měla ukládat jako tinyint(1). Je to z důvodu kompatability s předchozími verzemi databáze MySQL. Pro boolean hodnoty pak budou vygenerovány HTML prvky <input type="checkbox" … />, což označuje prostý zaškrtávací box. Generátor je možné předělat i tak, aby pro tyto hodnoty generoval prvky <select> s uvedením hodnot do prvků <option>. Je to na volbě každého programátory, s jakými prvky se mu lépe pracuje. Konstrukce formulářů by však měla být přizpůsobena především uživateli, který jej bude používat.

Prvek <select> využije generátor ještě v jiném případě. V databázi je možné mít atributy typu **enum**. Tyto atributy definují množinu povolených hodnot. V případě, že takový atribut v databázi

existuje, projeví se to v metadatech s názvem Type. Generátor tato data rozparsuje a vygeneruje prvek <select>, který bude obsahovyt všechny povolené hodnoty. Je-li navís atribut definován jako NOT NULL, přidá generátor do tagu <select> i volbu prázdné hodnoty.

Demonstraci generátoru je možné předvést na testovací tabulce 'xtest', která má strukturu:

```
CREATE TABLE `xtest` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `login` varchar(30) collate cp1250_czech_cs NOT NULL,
  `date_change` datetime NOT NULL,
  `type` enum('a','b','c') collate cp1250_czech_cs default NULL,
  PRIMARY KEY  (`id`),
  UNIQUE KEY `login` (`login`)
);
```

Vygenerovaný modul formuláře bude mít dle zkušební definice takový zápis:

```
/* MODULE GENERATOR - FORM STRUCTURE */
/* Database table name: xtest */
$form = new FormDeploy('POST', '', '', FormDeploy::FD_DEFAULT);
$form -> setFormLegend('Form label');
$form -> setFormInputSizes(25, 40);
$form -> addField (new textInput('submit', '', 'hidden'));
$form -> addField (new textInput('login', '', 'text', 30));
$form -> addField (new textInput('date_change', '', 'text', 10));
$form -> addField (new selectInput('type', array('a' => 'a', 'b' =>
'b', 'c' => 'c'), '', array('NULL' => 'NULL')))
$form -> addField (new submitInput('send', 'Uložit'));
$form -> render();
```

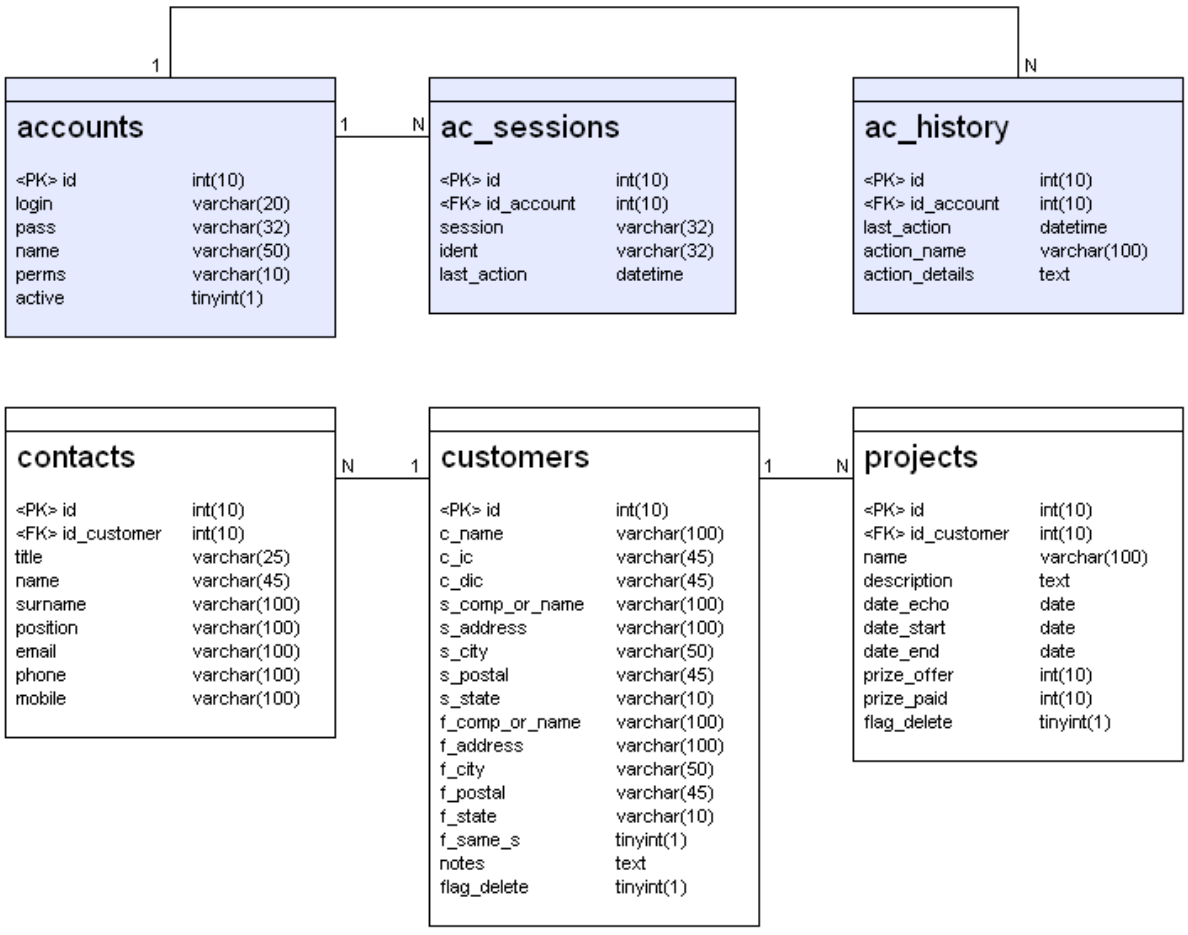
Je vidět, že všechny atributy databáze byly převedeny do zápisu HTML formuláře s využitím knihovny *FormDeploy*. Nyní stačí jemné doladění atributů a zápis lze hned použít. Přesný popis knihovny FormDeploy naleznete v příloze.

4 Vzorový administrační systém

Nejlepší demonstrací frameworku je vždy ukázka vytvořeného systému. Pro ukázku základních knihoven frameworku jsem zvolil jednoduchý model administrace firemního intranetu, který zahrnuje informace o zákaznících, kontaktech na osoby a správu projektů. Pro vygenerování kostry a formulářových objektů modulů byl použit vytvořený generátor. Je to klasická koncepce informačního systému se složitějšími databázovými vazbami pro ukázku interakce modulů.

4.1 Návrh systému

Tento zabezpečený systém bude využívat pokročilé techniky k zabezpečení autentifikace podle doporučení, popsaných v předchozí kapitole. Logování událostí bude pouze pro potřeby zjištění informací o několika posledních přihlášení do systému. Databáze bude sestávat ze 3 hlavních tabulek s vazbami 1:N podle následujícího ER-diagramu (tabulky pro logování informací a uchování záznamů pro potřeby přihlášení jsou vyznačeny barevně):



Obrázek 12 – ER-diagram vzorové aplikace

Jak vidno, oba dva vztahy mezi třemi projektovými tabulkami jsou typu 1:N. Pro každou z nich bude vytvořen jeden obslužný modul.

Zobrazování zákazníku bude sestávat z interního pojmenování zákazníka, adresy sídla a fakturační adresy, je-li jiná, než adresa sídla. Bude možné zobrazit všechny položky v podrobném detailu zákazníka. K dispozici budou všechny kontakty zákazníka viditelné pohromadě. Zobrazeno bude také to, kolik má zákazník kontaktů a projektů. Kontakty budou řazeny podle příjmení a jména. Pro projekty bude postačovat, když budou seřazeny podle data zápisu, dodatečné atributy budou pouze demonstrovat funkci generátoru modulů a knihovny pro vytváření pohledů na tabulku.

4.2 Systémové prvky

Není cílem tohoto frameworku nutit uživatele k používání již předprogramovaných konstrukcí. Každý má svůj osobitý styl, jaký preferuje při návrhu systému. Volnost a nezávislost jsou hlavními rysy frameworku Liquid Admin. Při dodržování obecně známých bezpečnostních postupů lze používat knihovny frameworku naprosto samostatně. Každá knihovna je vybavena vstupními a výstupními body.

Já jsem pro vývoj demonstračního projektu zvolil konstrukci, kde existuje společná prezentační šablona vrstvy view. Přihlášení do systému je vytvořeno klasickým stylem. Uživateli se zobrazí přihlašovací formulář, jako na obrázku 13:

Obrázek 13 – Přihlašovací formulář

Při odeslání tohoto formuláře projdou zadaná data validacemi a předají se autentifikační třídě. To je nejpodstatnější část při prvním přihlášení. Autentifikační třída nastaví session, přidá uživatelem definované kontroly a případny timeout nečinnosti a provede zápis záznamu do tabulky, kde se sessions uchovávají. Připomínám, že je na každém programátorovi, jak si přihlašování nastaví. Ve frameworku existuje demonstrační příklad, který vysvětluje přednastavený způsob autentifikace.

Po provedení tohoto prvního prověření uživatele se předává řízení navigátoru, který přesměruje uživatele na první systémový modul. Tímto modulem bývá většinou informační modul, kde je možné vidět informace o účtu, systémové poznámky a provádět změny nastavení účtu.

4.3 Moduly systému

Konstrukce každého modulu systému vychází z návrhu tak, jak bylo popsáno v kapitole 3.2.2 Struktura modulu. Na začátku modulu se provede načtení konfigurace a potřebných knihoven. Vytvoří se instance databázové třídy a jelikož ta je potřebná v procesu autentifikace, je možné ověřit uživatele. Ověření uživatele má mírně pozměněné funkční body než je tomu u přihlášení. Slouží k tomu metoda *check_auth()*. Markantní změna oproti funkci *login()* je v tom, že při ověřování se již nepracuje s tabulkou účtů, nýbrž pouze s tabulkou, kde jsou uloženy záznamy o session. Tabulka s účty je potřebná jen v bodě, kdy zjišťujeme informace o jménu a loginu přihlášeného uživatele a hlavně v případě, kdy prověřujeme uživatelská oprávnění.

Každý modul má také společnou šablonu pro zobrazení uživatelského rozhraní. Toto rozhraní je definováno ve složce *tpl/* soubor *admin_gui.tpl* a je zobecněno tak, aby pokrývalo potřeby všech modulů systému. Pro aplikaci šablony byl vytvořen skript, který se na konci každého modulu zavolá, předají se mu parametry pro generování a výstupem bude XHTML kód zobrazené stránky vrstvy view.

4.3.1 Modul zákazníci (customers)

Jednou z účinných knihoven frameworku je *StorageSession* (návrhový vzor singleton). Tento modul ji bude využívat k uchovávání informací ke stránkování záznamů, které bude modul zobrazovat. StorageSession může ukládat pouze do svého jmenného prostoru (namespace), tedy je nutné jej specifikovat při vytváření instance. Jako napespace identifikátor bude sloužit například název modulu. Je to snadno zapamatovatelné a intuitivní.

Pro vytvoření struktury modulu jsem použil generátor. Do vygenerované struktury je nutné doplnit akce, jaké budou v modulu dostupné a poté lze přistoupit k implementaci obslužných metod. Akce v modulu mohou být volné a vázané. Volné akce jsou takové, které nepotřebují dodatečné parametry, které jsou závislé na datech v databázi. Vázané akce jsou ty, jenž se váží na nějaký záznam v databázi. Většinou je vázaným akcím nutné předat nejméně parametr *id*, který tyto akce spojí s konkrétním záznamem. V modulu budou dostupná jediná volná akce – **zobrazit seznam zákazníků**. Vázané akce budou - **detail**, **kontakty**, **projekty**, **editovat**, **smazat**.

Programování modulu začíná vytvořením metody, která zobrazí data. Taková metoda bude zároveň implicitně volaná při zobrazení prvního výstupu modulu. Zobrazení dat je velmi jednoduché , použijeme-li k tomu knihovnu *MySQLDataTable*. Při dotatečné specifikaci SQL dotazu bude z výstupu vygenerovaná tabulka, jako je tomu v následující ukázce:

```
$db -> query("SELECT c.id, c.c_name as `Označení`,
CONCAT(IFNULL(CONCAT('IČ: ',c.c_ic),''),'\n',IFNULL(CONCAT('DIČ:
',c.c_dic),'')) as `IČ, DIČ`,
```

```
IF(c.pdf_invoice=1,'Ano','Ne') as `PDF`, c.pdf_email as `E-mail
(faktury)`,
CONCAT(COUNT(o.id),' kontakty','\n',COUNT(p.id),' projekty') as
`Informace`
FROM customers c
LEFT JOIN contacts o ON (c.id = o.id_customer)
LEFT JOIN projects p ON (c.id = p.id_customer)
WHERE (c.flag_delete = 0)
GROUP BY c.id
ORDER BY c.c_name" . $lister -> limit());
```

Lze vidět, že na konci kódu je volána metoda *limit()* knihovny *Lister*. Knihovnu lister lze využít ke stránkování výpisu. Metoda *limit()* vygeneruje do SQL dotazu podle předaných parametrů zápis LIMIT from,count. Pro vytvoření stránkování je pak nutné provést obdobný kód:

```
$lister = new Lister('list');
$lister -> url(SYSTEM_URL, PAGEFILE)
        -> classname('lister')
        -> jumps(true)
        -> total($db -> numRows())
        -> items(10)
        -> range(3)
        -> page($stor -> read('lister_page'))
        -> render();
```

Při předání SQL dotazu ke zpracování knihovně MySQLDataTable a dpolnění akcí je výstupem obdobná tabulka (zde nastylovaná pomocí CSS):

Záznam **1** - **10** z **18** Strana **1** 2 » »»

id	Označení	IČ, DIČ	PDF	E-mail (faktury)	Informace					
1	Amarost	IČ: 26888416 DIČ: CZ26888416	Ano		0 kontakty 0 projekty	Detail	Kontakty	Projekty	Editovat	Smazat
18	Frenk		Ano	frenk@cmail.cz	0 kontakty 0 projekty	Detail	Kontakty	Projekty	Editovat	Smazat
11	ITUM	IČ: 60371749	Ano		0 kontakty 0 projekty	Detail	Kontakty	Projekty	Editovat	Smazat

Záznam **1** - **10** z **18** Strana **1** 2 » »»

Obrázek 14 – Tabulka s daty z databáze pro modul zákazníci

Pohled na data je vytvořen a nyní lze přistoupit k tvorbě další důležité obslužné metody. Ta bude obsahovat formulář pro vkládání dat do databáze a pro usnadnění bude opět použit generátor, který pomůže s návrhem formuláře. Funkce generátoru je podrobně popsána v kapitole 3.4 Generátor modulů. Obdobným způsobem budou vytvořeny i další metody, jako editace a odstranění záznamů. Editační formulář bývá do značné míry podobný s editačním. Je na volbě programátora, zvolí-li pro editaci jinou metodu, než pro vkládání, nebo zda tyto metody sjednotí a výběr správné akce

rozliší parametry. Pro udržení přehlednosti kódu doporučuji vytvářet pro jednu akci vždy samostatnou metodu.

Přichází na řadu propojení modulu s modulem kontakty. Jak vidno, byly definovány dvě akce, které logicky přísluší jiným modulům. Jedná se o zobrazení kontaktů a projektů vybraného zákazníka. Při konstrukci modulů je striktně dodržováno pravidlo nemíchat obsluhu více tabulek do jednoho modulu, a tedy je nutné moduly propojit a obsluhu definovat v modulu kontakty.

Uvědomíme-li si, že máme k dispozici knihovnu *StorageSession*, která prostřednictvím obsluhy session parametrů může definovat stav aplikace, je řešení propojení zřejmé. Modul kontakty zobrazuje pouze kontakty k nějakému zákazníkovi. Nelze zobrazit kontakty, pokud neznáme zákazníka. Identifikace zákazníka je pochopitelně jeho atribut *id* v databázi. Metoda pro zobrazení kontaktů tedy použije *StorageSession*, změní jmenný prostor na modul kontakty, a uloží *id* zákazníka, jehož kontakty jsou žádány. Probíhá přesměrování do modulu kontakty, vyvolání uloženého *id* a výběr dat z databáze. Samozřejmostí je ošetření vstupních a výstupních parametrů, které by mohly vést k bezpečnostním rizikům v aplikaci.

Zajímavostí je, že modul kontakty nemusí v hlavním menu systému být vůbec uveden, protože uživatel se dostane do modulu kontakty a tedy i na jeho akce pouze přes modul zákazníci. Při zavádění tohoto modulu tedy definujeme, že je podřízený modulu zákazníci a systém tento modul nebude zobrazovat v hlavním menu. O zavádění modulů bude pojednáno v kapitole 4.3.3 Zavádění modulů.

4.3.2 Moduly kontakty (contacts) a projekty (projects)

Tvorba modulu je proces opakování stejných postupů, jako při tvorbě prvního modulu systému. Podrobnosti o tvorbě těchto dvou modulů zde tedy nebudou rozepsány podrobně. Zájemce si může pro získání podrobných informací o modulu zobrazit zdrojový kód demonstrační aplikace.

Zbývající dva moduly jsou definovány jako podřízené modulu zákazníci. Protože není možné dostat se do těchto modulů z hlavního menu, měl by mít každý z těchto modulů definovanou jednu akci pro návrat do předcházezejícího modulu, zkrátka tam, odkud byl uživatel nasměrován.

Stejným způsobem, jako modul kontakty, je vytvořen i modul projekty. Jelikož v těchto dvou modulech platí stejná závislost i databázové vztahy, je i struktura do značné míry stejná. Pouze zpracování dat probíhá nad jinou databázovou tabulkou s jinými atributy.

4.3.3 Zavedení modulů

Poslední a velmi snadný krok je zavedení hotových modulů do systému. To se provádí definicí inicializačních parametrů modulů v souboru *module_init.php.*

```
$MODULES['mod.customers.php'] = array(
    'menu' => 'Zákazníci', 'title' => 'Zákazníci', 'perms' => 'A',
    'submodules' => array('mod.contacts.php', 'mod.projects.php')
```

```
);

$MODULES['mod.contacts.php'] = array(
    'title' => 'Kontakty', 'perms' => 'A'
);

$MODULES['mod.projects.php'] = array(
    'title' => 'Projekty', 'perms' => 'A'
);
```

Jak je vidět v zápisu, modul může mít libovolný pošet podřazených modulů. Dokonce i podřazený modul může mít další sobě podřazené moduly. Lze tedy definovat velmi robustní strukturu závislostí modulů. Pouze je třeba dávat pozor na smyčky. Systému by to nejspíše neuškodilo, ale ovládání by nemuselo být zrovna optimální.

Administrační systém, který je nyní téměř hotový, bude sloužit převážně k demonstračním účelům frameworku. V této chvíli je čas na spuštění ukázkové aplikace.

4.4 Instalace a testování

Webové administrační systémy lze v podstatě nainstalovat vytvořením databáze a založením potřebných tabulek. V konfiguračním souboru poté stačí definovat přístupové údaje k databázi a některé další direktivy, jako maximální velikost odesílaných souborů a adresu systému (je-li to třeba). Operuje-li administrační systém se soubory, pak by měly být příslušné adresáře nastaveny pro možnost zápisu. Posledním prvkem instalace je vytvoření uživatelských účtů. Je-li v aplikaci vytvořen modul správy uživatelů, minimálně první účet bude třeba vložit do databáze ručně. V případě správy uživatelů by měl tento modul pamatovat na to, aby neodstranil všechny účty. Pak by se nebylo možné k aplikaci přihlásit a byl by nutný opětovný ruční zásah do databáze.

Nikdo není bezchybný a i nejlepší programátor udělá časem chybu. Testování aplikace je tedy neoodělitelnou součástí vývoje i instalace. Systém musí přijímat správné vstupy a zobrazovat správná data ve vhodném uspořádání. Nemalou část procesu testování zabírá kontrola bezpečnosti. Je nutná odolnost vůči SQL injekcím a s tím souvisí správné zpracování požadavků od uživatelů.

Při návrhu a následné validaci formulářů se často zapomíná na kontrolu některých prvků, jako checkbox nebo selectbox. To, že uživatel nemůže do formuláře zapsat svá data a je nucen použít přednastavená není zárukou, že do aplikace nikdy nedojdou nepředpokládaná data. Zkušení uživatelé mohou poslat svůj vlastní POST požadavek v přihlášeném účtu a pokud aplikace předpokládá vstupní hodnotu číslo, pak by mohlo dojít k nečekané události, pokud by parametrem byl text. Podceňování uživatelů je špatnou strategií.

5 Nasazení a provoz administračních systémů na frameworku Liquid Admin

Nejlepší odezvu na kvalitu vytvořeného systému lze získat praktickým nasazením v ostrém provozu. Jelikož framework Liquid Admin byl z části vyvinut již v roce 2006, bylo v něm vytvořeno několik aplikací. S minimálně roční odezvou lze tedy zjistit a zhodnotit, jak efektivně byla celá technologie použita. Ranné verze frameworku měly značně pozměněnou architekturu a při vývoji prodělaly téměř všechny knihovny refaktorizaci. Základní idea frameworku byla však stejná. Podařilo se vyvinout několik systémů, které nelze vzhledem k nasazení považovat za elementární. Zhodnocení dvou takových systémů bude předmětem této kapitoly.

5.1 Provozní prostředí

Shodou okolností jsou oba systémy provozovány na vlastních a samostatných serverech. To přináší hlavně výhodu v bezpečnosti. Ve výkonu aplikací se to nijak zásadně neprojeví, jelikož framework je nenáročný. Provoz na serveru je však výhodnější převážně proto, že je možné server nakonfigurovat libovolným způsobem a aplikace tedy není omezena, jak by tomu mohlo být v případě umístění na veřejný hosting.

V obou případech je jako server použit Apache, interpret PHP 5 a databáze MySQL verze 5. Systémy jsou vystaveny na veřejné internetové adrese. K jednomu z nich vede odkaz z vebové prezentace, jelikož slouží i jako klientský systém. Bylo zvoleno optimální zabezpačení i pro uživatele, kteří se do systému mohou přihlašovat z podnikových sítí, a tedy vystavují svůj účet dalším rizikům.

5.2 Zhodnocení provozu – Megacars.cz

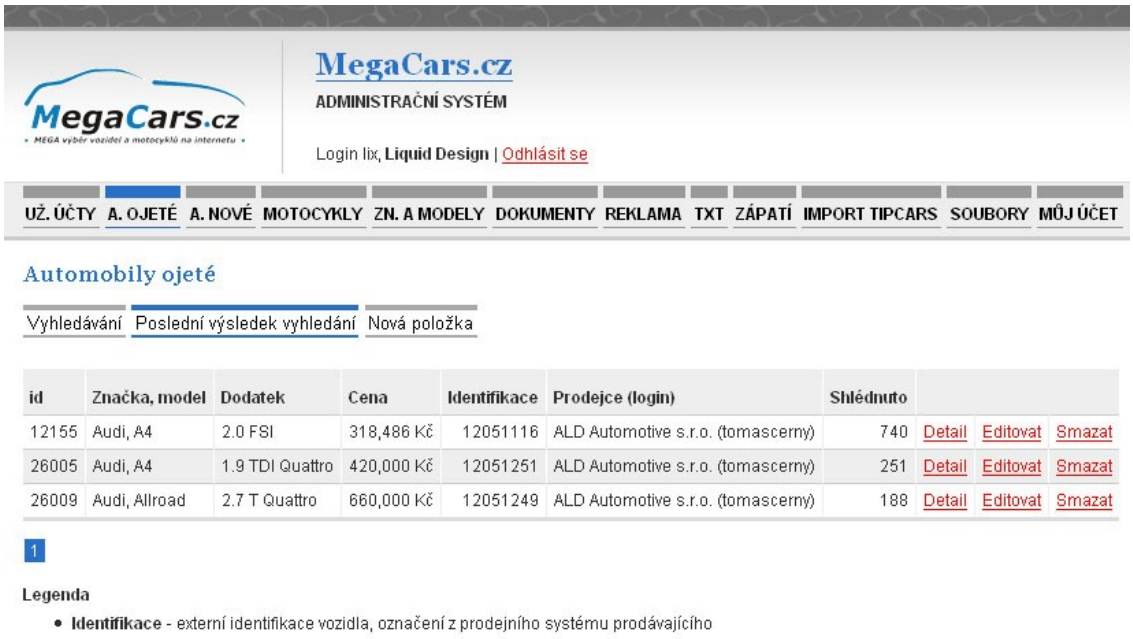
Prvním z porovnávaných systémů je Magacars.cz. Ve frameworku Liquid Admin má tento systém vytvořenou aplikaci administrace i klientského systému. Systém zahrnuje 4 role:

- **superadmin** - servisní účet – zákazníkovi nedostupný
- **admin** - správa systému megacars.cz
- **firma** - uživatelský účet typu firma
- **osoba** - uživatelský účet typu osoba

Pro zhodnocení aplikace bude mít asi největší význam administrátorská role, jelikož klientská sekce je omezená pouze na správu jednoho účtu. Administrátor oproti tomu může spravovat všechna data systému a všechny účty.

V době kontroly systému obsahoval přes 1100 uživatelských účtů. Tyto účty je možné řadit, filtrovat a následně kontrolovat podle data založení, typu účtu a expirace účtu. Práce s uživatelskými účty je komfortní a poměrně rychlá.

Systém Megacars.cz ze zabývá prodejem nových a ojetých vozidel a motocyklů a v současné době je v systému evidováno přes 12000 ojetých 300 nových vozidel a rovněž asi 300 motocyklů. Do administrace byl implementován modul, který dokáže vyhledávat a řadit vozidla podle prodejce, značky, modelu, roku výroby a interní identifikace. Vyhledaná vozidla jsou zobrazena v tabulce na několik stranách dle potřeby a každý záznam má k dispozici akce, jako detail, editaci a smazání. Při mazání je mnohdy zasahováno až do pěti tabulek a do různých adresářů s fotografiemi a soubory. Přesto je mazání velmi rychlé a systém odolný proti chybám v důsledku možného přerušeného mazání některých souborů.



Obrázek 15 – Náhled na informační systém Megacars.cz

Jako další modul obsahuje administrace souborový organizátor. Nově přidaným moduem byl i XML import dat z jiných serverů, jako Tipcars.com či prodejních systému společnosti Teas. XML soubory dosahují mnohdy i velikosti 50MB, jelikož mohou obsahovat fotografie v několika velikostech. Zbytek modulů slouží k ovládání webové prezentace a není tedy takového rozsahu, jako předchozí zmíněné.

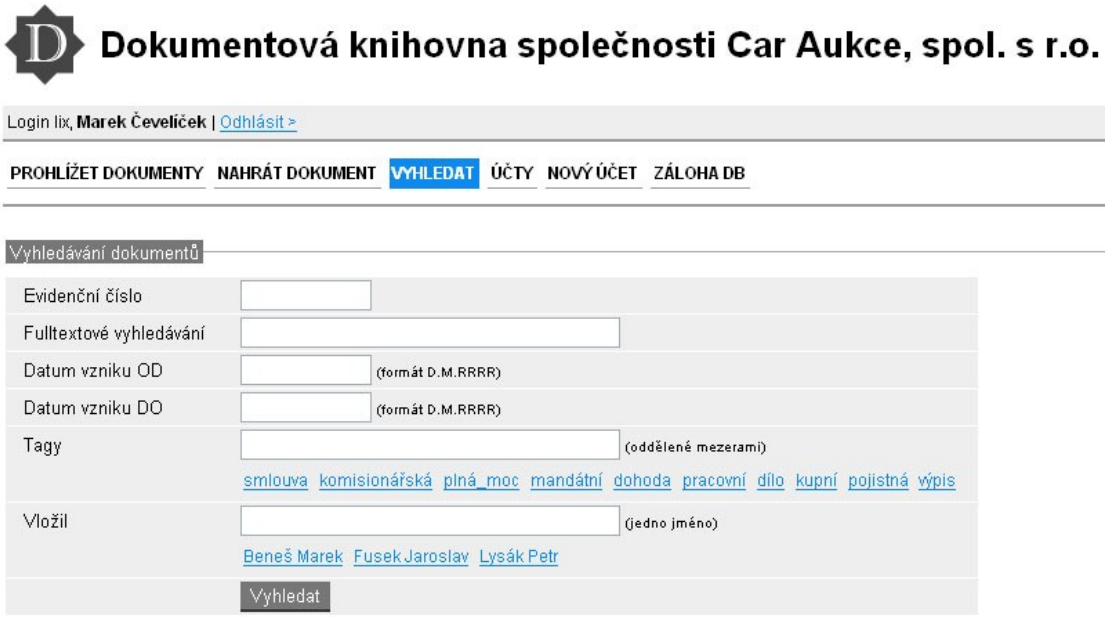
Uživatelské prostředí je přívětivé a prozatím nebyly evidovány žádné žádosti o vylepšení funkcí nebo ovládání aplikace. Lze tedy předpokládat, že systém je dostatečně intuitivní. Jednou z nevýhod systému je, že vyšší zabezpečení přihlášení je pro každého uživatele definováno

administrátorem, tedy uživatel sám se nemůže rozhodnout, jaký způsob v danou chvíli zvolí. Při kontrole systému se objevilo několik málo nekonzistencí v databázi, které byly způsobeny nejspíše při odstraňování záznamů a následné chybě na serveru. Žádná z těchto nekonzistencí není však pro běh systému limitující a lze je snadno dohledat a opravit kontrolním skriptem.

5.3 Zhodnocení provozu – dokumentová knihovna, http://aukcevozidel.cz/doclibrary/

Před několika měsíci byla vytvořena v téměř poslední verzi frameworku aplikace, která měla za úkol udržovat databázi dokumentů. Tato práce byla provedena na zakázku a v současnosti je provozována na několika serverech. Hlavní objem dat tvoří rozsáhlé smlouvy, obchodní a úřední doklady a kupní dokumenty. Vše souvisí s prodejem vozidel a je vysoce důležité, aby k dokumentu měl přístup pouze oprávněný uživatel.

Systém je používán spolupracujícími firmami, které řídí aukce a obchod s vozidly. Rozdělení rolí systému je tedy jednodušší – **admin** a **uživatel**. Administrátor má přehled o všech dokumentech. Uživateli je přiřazen stupeň utajení a každý dokument má rovněž stupeň utajení – číslo od 1 do 3. Každý uživatel pak může zobrazit a stahovat pouze dokumenty, které mají stejný nebo nižší stupeň utajení. Uživateli je dále umožněno vyhledávání dokumentů a nastavení svého účtu, včetně změny hesla. Administrátorské funkce obsahují navíc nahrávání dokumentu, správu všech účtů a export databáze.



Obrázek 16– Náhled na informační systém dokumentové knihovny

Aplikace obsahuje poměrně nízký počet účtů – cca 10 a současné množství dokumentů je asi 150. Dokumenty je možné tagovat (přiřazovat jim klíčová slova). K tomuto účelu byla vyvinuta speciální knihovna v JavaScriptu. Napojení a používání vlastních knihoven ve frameworku je jednoduché a tedy implementace byla snadná. Systém zahrnuje přesné a do značné míry nastavitelné fultextové vyhledávání v dokumentech.

Za dobu několika měsíců, kdy je systém aktivně používán byly kladné ohlasy především na zpracování vyhledávacího modulu, který dokáže filtrovat zobrazení dokumentů a udržovat nastavení filtru (pomocí knihovny *StorageSession*). Komfortní je také zadávání tagů k dokumentům. Systém inteligentně vybírá nejčastěji používané tagy a tyto nabízí k vyplnění nejdříve. Klikací rozhraní šetří čas před ručním psaním tagů a eliminuje výskyt chybně tagovaných dokumentů.

Problematické bylo zpracování skriptu, který zprostředkovává stahování dokumentů a implementuje kontrolu oprávnění a stupeň utajení. Internetové prohlížeče nejsou ve zpracování bufferovaného výstupu klientovi příliš jednotné a každý se chová trochu odlišně. Za pomocí zvolení správných HTTP hlaviček bylo dosaženo dostatečné úrovně bezpečnosti. Dokukumenty je třeba chránit s nejvyšší prioritou, jelikož mohou obsahovat osobní údaje.

Celkově je systém z mé strany i ze strany objednavatele projektu hodnocen velmi pozitivně. Nápadů na vylepšení je minimum a tuto práci lze označit za úspěšnou.

6 Pokračování projektu

Všechny aplikace časem stárnou, nejsou-li aktualizovány a dále vyvíjeny. Každým rokem přibývá mnoho nových technologií a jsou objevovány stále pokročilejší postupy, jak vytvořit výslednou aplikaci rychleji, levněji a lépe zpracovanou. Neudržovaný projekt se již během několika málo let stáva nepoužitelným a zaostávajícím za moderními a vyvíjenými projekty. V tomto ohledu jsou na špičce aktuálnosti open source projekty. K takovým existuje komunita lidí, kteří se o projekt starají. Většina open source projektů není výdělečná a je spíše otázkou zájmu, jak se na vývoji podílejí příznivci těchto projektů.

Cílem tohoto projektu bylo vytvořit administrační framework. Pokračováním projektu je jeho údržba a vývoj. Základní knihovny by mohly být doplněny o další potřebné rozšíření. Kupříkladu atraktivním se stává protokol SOAP, který umožňuje výměnu zpráv založených na jazyku XML přes počítačové sítě (tedy i internetovou síť prostřednictvím protokolu TCP/IP). Jazyk WSDL (Web Services Description Language) umožňuje definovat strukturu zpráv, které tak mohou putovat nezávisle na programovacím jazyku z jednoho informačního systému do druhého. Takovou knihovnu, která by podporovala SOAP a WSDL však ve frameworku Liquid Admin nenajdete. Je nutné vyvinout iniciativu a framework rozšířit o nové knihovny, které jsou v administračních systémech žádané a potřebné.

Velké pokroky by mohlo přinést vyvíjení generátoru modulů. Současný stav je spíše ukázkový. Rozhodně však není možné použít generátor na všechny případy modulů. Při rozšíření generátoru by jako jedna alternativa mohla být klikací prostředí, které umožní tvorbu modulů do administračních systémů mnohem pohodlnějším způsobem. Takové prostředí by bylo zároveň odladěné, a tedy by výsledné moduly neobsahovaly chyby a zbytečné bezpečnostní trhliny, které mohou být způsobeny nepozorností programátora nebo nedostatečnou analýzou systému. Generátor, který by byl tak dokonalý, by mohl být obsahem celé jedné diplomové práce. Generování programu je zajisté složitý a obtížný úkol.

V každém projektu je vždy co zlepšovat. Při návrhu těchto vylepšení je třeba věnovat dostatek času analýze a myslet na budoucí trendy, nové technologie a hlavně na uživatele, kteří budou výsledek používat.

7 Závěr

V procesu analýzy aplikace jsem zformuloval obecné požadavky uživatelů na administrační systém. Při návrhu jsem zohlednil mnoho svých zkušeností s podobnými systémy a zároveň si uvědomil některá nepoznaná úskalí tvorby návrhů systémů. Podařilo se mi realizovat vhodnou strukturu kódu, který je flexibilní a přehledný. V průběhu tvorby projektu jsem objevil některé netušené nedostatky programovacího jazyka PHP, které se projevily převážně v pokusech o vytvoření dostatečně bezpečného přihlašování uživatelů s využitím sessions.

Podařilo se mi realizovat systém, který zohledňuje principy přístupnosti a použitelnosti internetových aplikací. Realizace je předmětem mého diplomového projektu. Způsob, jakým jsou v systému například generovány tabulky s databázovými daty považuji za unikátní a nikde jinde jsem se s podobnou implementací doposud nesetkal. Jedná se vlastně o vytvoření univerzálního modelu pro většinu dat, jenž je třeba zobrazit v uživatelském rozhraní. Tento model je schopen zároveň na dodaných datech vygenerovat pro uživatele pohled (view), který lze využitím několika metod velmi flexibilně nastavit. Odpadá tedy práce s tvorbou odlišných modelů a pohledů pro každý druh dat, která zobrazujeme uživateli.

Jedním z přínosů tohoto projektu je názorná ukázka využití jednoduchých technik k dosažení požadovaného výsledku. Zpracování principů, kterými by se měla správně zabezpečit webová on-line aplikace je podstatnou součástí této práce. Bezpečnost hraje v administračních systémech velkou roli a úskalí této problematiky jsou velmi rozsáhlá. Všechny své poznatky jsem se pokusil v této práci shrnout, poskytnout jejich kompletní přehled a rovněž nabídnout řešení lepšího zabezpečení.

Při realizaci diplomového projektu bylo hlavním úkolem využít zde uvedené teoretické poznatky k návrhu a programování univerzálního frameworku, který bude možné použít pro tvorbu administračních systémů většiny webových projektů. Takto vytvořené administrační systémy by měly být snadno rozšiřitelné a pohodlně ovladatelné uživateli.

Každá práce může být dále rozšířena a posunuta na novou úroveň znalostí. Můj projekt není výjimkou a rád bych v budoucnu na své výsledky navázal a pokusil se o distribuci tohoto frameworku jako open source projektu.

7.1 Přínos projektu

Hlavním cílem projektu bylo vytvoření frameworku, který by obsahoval dostatečně použitelné knihovny, aby pomocí nich bylo možné realizovat většinu administračních systémů. Framework má modulovou strukturu a vytvořené systémy jsou velmi přehledné. I bez programového manuálu by měl být nezasvěcený programátor schopen rozhodnout k čemu jaká část modulu slouží a rychle se zorientovat v cizím kódu. Tvorba administrací pomocí tohoto nástroje je velmi rychlá a mnohem

bezpečnější, než by tomu bylo při vývoji bez použití připravených knihoven. Všechny knihovny byly navrženy s ohledem na intuitivní použití, vysokou bezpečnost, možnost uživatelského nastavení a v neposlední řadě jsou připraveny pro libovolné rozšíření funkčnosti.

Částečně se projekt zabýval i možnostmi, jak by bylo realizovatelné automatické generování částí modulů do administrace. Z této úvahy vzešel nástroj – generátor modulů (a formulářových objektů). Koncepce generátoru byla postavena na generování kódu z popisu SQL tabulek. Použitím generátoru se ještě více podařila urychlit konstrukce modulů. Generátor modulů je oblast, v níž by framework mohl doznat ještě mnoha vylepšení. Zpracování základní myšlenky se však povedlo bez výrazných potíží a výstupy jsou použitelné.

Literatura

[1] Gilfillan, I.: Myslíme v jazyku MySQL 4. Grada Publishing, Praha, 2003.

[2] Gutmans, A., Saether Bakken, S., Rethans, D.: Mistrovství v PHP 5. CP Books, Brno, 2005.

[3] Kosek, J.: PHP – tvorba interaktivních internetových aplikací. Grada Publishing, Praha, 1999.

[4] Pošmura, V.: Apache příručka správce WWW serveru, Computer Press, Praha, 2002.

[5] Hernandez, J. M.: Návrh databází, Grada Publishing, Praha, 2005.

[6] Druska, P.: CSS a XHTML, Grada Publishing, Praha, 2006.

[7] Holzner, S.: Mistrovství v AJAXu, Computer Press, Praha, 2007.

[8] Semecký, J.: Autorizace uživatelů v PHP. Interval.cz, 2001.
URL: http://interval.cz/clanky/autorizace-uzivatelu-v-php/.

[9] Wikipedie: Model-view-controller. Wikipedie.
URL: http://cs.wikipedia.org/wiki/Model-view-controller.

[10] Vrána, J.: Zabezpečení session proměnných, PHP triky, 2005.
URL: http://php.vrana.cz/zabezpeceni-session-promennych.php

[11] Růžička, P.: Bezpečnost především - cross-site skripting a session-stealing. Interval.cz, 2002.
URL: http://interval.cz/clanky/bezpecnost-predevsim-cross-site-skripting-a-session-stealing/.

[12] Wikipedia: Design pattern (computer science). Wikipedia.
URL: http://en.wikipedia.org/wiki/Design_pattern_%28computer_science%29/.

[13] Wikipedia: Cross-site scripting. Wikipedia.
URL: http://en.wikipedia.org/wiki/Cross-site_Scripting

[14] Wikipedia: Session poisoning. Wikipedia.
URL: http://en.wikipedia.org/wiki/Session_poisoning

[15] The Open Web Application Security Project (OWASP): Session Management. The OWASP.
URL: http://www.owasp.org/index.php/Session_Management

[16] IETF Tools: The BSD syslog Protocol. IETF Tools.
URL: http://tools.ietf.org/html/rfc3164

Seznam příloh

Příloha 1. Knihovny frameworku

Příloha 2. SQL tabulek knihovny Auth

Příloha 1 – Knihovny frameworku

ActionController

konstruktor (string $action_param)

Vytvoření instance třídy a definice parametru s názvem akce.

metoda addAction (string $title, string $param) : void

Přidá novou akci do modulu. Název akce $title a předávaná hodnota parametru $param.

metoda authorize (string $allowed, string $owned) : boolean

Autorizace uživatele pro spuštění požadované akce. Vlastní práva $owned a povolená $allowed.

metoda setActionCmd (string $string) : void

Nastavení / změna parametru s názvem akce.

metoda setAccessModule (boolean $bool) : void

Povolení / zakázání přístupu k celému modulu.

metoda setAccessFailMsg (string $msg) : void

Nastavení chybového hlášení při pokusu o neoprávněný přístup k modulu.

metoda isExplicitAccessControl () : boolean

Kontrola, zda je přístup k modulu určen podle oprávnění nebo explicitně programátorem.

metoda getActions () : array

Získání všech akcí modulu.

metoda getAccessModule () : boolean

Kontrola povolení o přístup k modulu.

metoda getAccessFailMsg () : string

Získání chybového hlášení při pokusu o neoprávněný přístup k modulu.

metoda getActionRequest () : string

Získání hodnoty parametru požadované akce.

Auth

konstruktor ()

Vytvoření instance třídy. Třída je vytvořena jako singleton (pouze jedna instance).

metoda getInstance () : Auth

Vrátí instanci třídy nebo vytvoří novou instanci, pokud ještě neexistuje.

metoda getSessionId () : string

Vrátí session identifikátor.

metoda authenticate () : void

Provede přípravu třídy pro autentifikaci a vytvoří session.

metoda timeout () : void

Nastaví timeout pro odhlášení. Timeout je ukládán do tabulky s záznamy o session.

metoda secret () : void

Nastaví, zda použít pokročilejší ověření uživatele podle IP adresy a prohlížeče.

metoda login () : void

První přihlášení uživatele do aplikace od vytvoření session.

metoda check_auth () : void

Ověření přihlášeného uživatele, když už je nastavena session.

metoda isValid () : boolean

Kontrola, zda je uživatel přihlášen. to samé, co kontrola shody s konstantou Auth::SUCCESS.

metoda getIdentity () : integer

Získání jedinečného identifikátoru uživatele z databáze. Zpravidla sloupec *id*.

metoda getStatus () : integer

Kontrola stavu přihlášení / ověření uživatele. Lze testovat na konstanty Auth::SUCCESS, Auth::FAILURE, Auth::FAILURE_CREDENTIAL_INVALID, Auth::FAILURE_TIMEOUT, Auth::FAILURE_IDENTITY_NOT_FOUND, Auth::FAILURE_IDENTITY_AMBIGUOUS.

metoda logout () : void

Odhlášení uživatele, zrušení session proměnných i samotné session.

DB

konstruktor (string $host, string $login, string $pw, string $dbname)

Vytvoření instance třídy.

metoda charset (string $charset) : void

Nastavení kódování pro komunikaci s databází.

metoda close () : void

Uzavření aktivního spojení.

metoda query (string $q) : void

Provedení SQL dotazu $q.

metoda getResult () : sql result

Získání výsledku provedeného SQL dotazu.

metoda insert (string $tbl, array $arr) : void

Vložení záznamu do tabulky $tbl. Hodnoty jsou předány jako pole array(attribute => value).

metoda insert_heap (string $tbl, array $a_keys, array $a_2dim_values) : void

Vložení několika záznamů hromadně v jednom INSERT příkazu. Atributy jsou uloženy v poli $a_keys a hodnoty ve dvourozměrném poli $a_2dim_values.

metoda update (string $tbl, array $arr, integer $row) : void

Provedení příkazu UPDATE nad tabulkou $tbl, parametry $arr. Podmínka pro specifikaci identifikátoru ID pro cílení dotazu je v $row.

metoda update_extend (string $tbl, array $arr, string $where_cond) : void

Provedení příkazu UPDATE nad tabulkou $tbl, parametry $arr. Obecná podmínka pro specifikaci dotazu je v $where_cond.

metoda fetchAll (integer $type) : array

Vrátí dvourozměrné pole s výsledkem celého SQL dotazu. Parametr $type určuje, zda bude hodnota uložena v poli indexovaném numericky nebo asociativně. Povolené hodnoty jsou DB::MYSQL_NUM, DB:MYSQL_ASSOC, DB:MYSQL_BOTH.

metoda fetchNext (integer $type) : array

Vrátí pole s výsledkem dalšího záznamu v pořadí od pozice kurzoru. Parametr $type určuje, zda bude hodnota uložena v poli indexovaném numericky nebo asociativně. Povolené hodnoty jsou DB::MYSQL_NUM, DB:MYSQL_ASSOC, DB:MYSQL_BOTH.

metoda fetchPrev () : array

Vrátí pole s výsledkem předchozího záznamu v pořadí od pozice kurzoru.

metoda fetchCurr () : array

Vrátí pole s výsledkem aktuálního záznamu v pořadí podle pozice kurzoru.

metoda getCurrPosition () : integer

Získání pozice kurzoru.

metoda numRows () : integer

Získání počtu vybraných záznamů SQL dotazu.

metoda affectedRows () : integer

Získání počtu ovlivněných záznamů operací UPDATE.

metoda lastId () : integer

Hodnota poslední vygenerované hodnoty auto increment.

metoda quoteInto ($stmt, $value, $type) : void

Obdobně jako query(), ale v SQL dotazu $stmt je nahrazen symbol ? hodnotou $value. Hodnota $value je navíc ošetřena podle typu předávané hodnoty specifikované v $type.

metoda quote ($value, $type = false) : string

Ošetření hodnoty $value podle typu předávané hodnoty specifikované v $type.

metoda getQueryTime () : integer

Získání času trvání provedení SQL dotazu.

FormDeploy

konstruktor (string $method, string $action, string $class, string $encoding)

Vytvoření instance třídy a nastavení parametrů $method, $action, $encoding. Dodatečně lze XHTML tagu <form> nastavit třídu $class.

method setFormLegend(string $legendtext) : void

Nastaví XHTML tag <legend> pro určení titulku formuláře.

method setFormInputSizes(integer $size) : void

Nastavení povolených šířek pro XHTML tagy <input>. K nastaveným šířkám se tyto tagy budou zarovnávat.

method setLabelNoteClass (string $label, string $note) : void

Nastavení atributu class pro elementy <label> a pro text s poznámkou.

method groupFields () : void

Seznam názvů formulářových elementů, které mají být vykresleny ve skupině na jednom řádku.

method addLabel (string $fieldname, string $labeltext) : void

Přidání štítku pro formulářový element s názvem $fieldname a textem $labeltext.

method addField (mixed $field) : void

Přidání objektu formulářového elementu.

method addNote (string $fieldname, string $notetext) : void

Přidání poznákmy pro formulářový element s názvem $fieldname a textem $labeltext.

method setErrors (array $array) : void

Nastavení chybových textů do formuláře.

method setValues (array $array) : void

Nastavení hodnot formulářovým prvkům.

method setValuesSQL (sql result $result) : void

Nastavení hodnot formulářovým prvkům z výsledku SQL dotazu.

method render () : void

Vykreslení hotového formuláře v XHTMl podobě.

FormDeploy / formulářové objekty

Objekt	Konstruktor
textInput	konstruktor ($name, $value, $type, $maxlen, $attr)
numberInput	konstruktor ($name, $value, $size, $attr)
boxInput	konstruktor ($name, $value, $type, $text, $checked, $attr)
selectInput	konstruktor ($name, $array, $value, $zero)
selectInputSQL	konstruktor ($name, $result, $optionvalue, $optiontext, $selected, $zero)
textarea	konstruktor ($name, $value, $cols, $rows, $attr)
fileInput	konstruktor ($name, $attr)
imageInput	konstruktor ($src, $attr)
submitInput	konstruktor ($name, $value, $attr)

FormValidator

konstruktor (string $method)

Vytvoří instanci třídy validátoru a nastaví ze které superglobální proměnné se budou kontrolovat data. Implicitně POST.

method vildCard (& $error, integer $min, integer $max) : void

Interní funkce, která umožňuje používání „divokých karet“ v textu chyb. Tyto konstanty se nahradí odpovídajícím výrazem.

method validText (string $field, integer $min, integer $max, string $error) : void

Kontroluje správnost textového vstupu podle vložené délky řetězce.

method validRegular (string $field, string $pattern, string $error) : void

Kontroluje správnost vstupu podle určeného regulárního výrazu.

method validNumber (string $field, integer $min, integer $max, string $error) : void

Kontroluje správnost číselného vstupu podle zadané hodnoty.

method validDate (string $field, string $mode, string $error) : void

Kontroluje správnost data podle zvoleného formátu EU nebo US.

method validEmail (string $field, string $error) : void

Kontroluje správnost zadané e-mailové adresy.

method validError (string $field, string $error) : void

Přidá do pole explicitně definovanou chybu.

method getUserErrors () : array

Vrátí asociativní pole se všemi vzniklými chybami. Klíčem pole jsou name atributy formulářových prvků.

method getNumErrors () : integer

Vrátí počet vzniklých chyb.

method getValidStatus () : boolean

Vrátí validační výsledek true / false.

MySQLDataTable

konstruktor (sql result $sql, string $table_class, string $even_class, string $odd_class)

Vytvoří nový objekt z výsledku SQL dotazu a nastaví atributy class elementu <table>.

method commands (string $id, array $cmds) : void

Přidá ke každému záznamu akce $cmds a spojí je s jejich jedinečným *id*, které se nachází ve sloupci $id. Akce $cmds mají tvar array('action_name' => 'Action description')

method setURL (string $url) : void

Nastaví URL pro vygenerování absolutních odkazů. Není-li URL specifikovány, jsou generovány relativní odkazy.

method maxLength (string $a, integer $n) : void

Atributu $a omezí na zobrazování maximálně $n znaků.

method maxLengthGlobal (integer $n) : void

Globální nastavení maximální délky všech atributů tabulky na $n znaků.

method lineBreaks (string $b, integer $len) : void

Řádky budou zalomeny podle zvoleného nastavení $b, které může nabývat hodnot 'real', 'wrap', 'both'. Při zvolení 'wrap' bude řádek zalomen na $len znaků.

method hideColumns (string $c) : void

Definice atributů, které se nemají zobrazovat, ale jsou pro výběr důležité. Například atribut *id.*

method noTableHead () : void

Hlavička tabulky s názvy sloupců nebude zobrazena.

method showNull () : void

Hodnoty NULL budou zobrazeny v tabulce jako text *null*.

method alignColumn (string $c, string $align) : void

Nastavení zarovnání sloupce s názvem $c.

method substitute (string $c, array $a) : void

Substituce hodnot ve sloupci $c podle předlohy v poli $a. Je-li například ve sloupci hodnota 4, pak při existenci pole array(4 => 'ano') bude namísto hodnoty 4 zobrazena hodnota 'ano'.

method imageColumn (string $col_name, string $path, string $onclick) : void

Definování sloupce $col_name jako odkazu na obrázek. Je možné nastavit cestu do adresáře s obrázky a přidat událost $onclick, která se provede při kliknutí na zobrazený odkaz – výhodné při otevírání obrázku do nového okna pomocí JavaScriptu.

method fileColumn (string $col_name, string $path) : void

Definování sloupce $col_name jako odkazu na soubor. Je možné nastavit cestu do adresáře se soubory.

method render () : void

Zobrazení tabulky s daty z SQL dotazu v XHTML formě podle nastavených vlastností.

method render_vertical () : void

Zobrazení prvního záznamu tabulky s daty z SQL dotazu v XHTML formě podle nastavených vlastností ve vertikální podobě (hlavička tabulky na levé straně).

StorageSession

konstruktor ()

Vytvoření instance třídy. Třída je vytvořena jako singleton (pouze jedna instance).

method getInstance () : StorageSession

Vrátí instanci třídy nebo vytvoří novou instanci, pokud ještě neexistuje.

method realm (string $r) : void

Nastaví jmenný prostor s názvem $r pro ukládání hodnot proměnných. Do tohoto prostoru mohou pak metody třídy implicitně přistupovat.

method write (array $array, integer $flag, integer $access) : void

Zapíše hodnoty pole $array do souboru session. Názvy proměnných budou stejné, jako jsou klíče pole $array. $flag může být StorageSession::WRITE_ALWAYS (implicitní) nebo StorageSession::WRITE_IF_IS_SET – zapíše hodnotu proměnné pouze není prázdná. Je-li nastaven parametr $access = StorageSession::ACCESS_GLOBAL, pak hledá místo pro zápis v globálním jmenném prostoru.

method read (string $k, integer $access) : mixed

Přečte hodnotu $k ze souboru session. Je-li nastaven parametr $access = StorageSession::ACCESS_GLOBAL, pak hledá proměnnou v globálním jmenném prostoru.

method isEmpty (string $k, integer $access) : boolean

Kontrola, zda je nastavena proměnná $k. Je-li nastaven parametr $access = StorageSession::ACCESS_GLOBAL, pak hledá proměnnou v globálním jmenném prostoru.

method clear (string $k, integer $access) : void

Zruší proměnnou $k. Je-li nastaven parametr $access = StorageSession::ACCESS_GLOBAL, pak hledá proměnnou v globálním jmenném prostoru.

Příloha 2 – SQL tabulek knihovny Auth

```
CREATE TABLE `accounts` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `login` varchar(20) NOT NULL,
  `pass` char(32) NOT NULL,
  `name` varchar(50) NOT NULL,
  `perms` varchar(10) default NULL,
  `active` tinyint(1) default '1'
  PRIMARY KEY  (`id`),
  UNIQUE KEY `login` (`login`)
) ENGINE=MyISAM;

CREATE TABLE `ac_sessions` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `id_account` int(10) unsigned NOT NULL,
  `session` char(32) NOT NULL,
  `ident` char(32) default NULL,
  `last_action` datetime NOT NULL,
  PRIMARY KEY  (`id`)
) ENGINE=MyISAM;

CREATE TABLE `ac_history` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `id_account` int(10) unsigned default NULL,
  `datetime` datetime NOT NULL,
  `action_name` varchar(100) NOT NULL,
  `action_details` text,
  PRIMARY KEY  (`id`)
) ENGINE=MyISAM;
```