České vysoké učení technické v Praze

Fakulta elektrotechnická

BAKALÁŘSKÁ PRÁCE

Internetové stránky VLSI skupiny

Praha 2006 **Martin Zatřepálek**

České vysoké učení technické v Praze
Fakulta elektrotechnická

Bakalářská práce

Internetové stránky VLSI skupiny

Martin Zatřepálek
Vedoucí práce: Ing. Petr Fišer

Studijní program: Elektrotechnika a informatika, strukturovaný, bakalářský

Obor: Výpočetní technika

červen 2006

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č.121/2000 Sb. , o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne ……. …….

Podpis

Anotace

Tato práce se zabývá návrhem a realizací internetových stránek pro VLSI skupinu. Analytická část se zabývá výběrem dostupných technologií a standardů, aby stránky splňovaly požadavky na přehlednost a použitelnost ve všech dnes používaných prohlížečích. Webové stránky jsou rozděleny na veřejnou a neveřejnou část. Obě části jsou implementovány ve vybraných technologiích (PostgreSQL a PHP). U veřejné části byl kladen největší důraz na bezpečnost a odolnost aplikace vůči případným pokusům o útok. V této části členové VLSI skupiny prezentují projekty pro studenty, novinky, které se ve skupině udály, a publikace, které skupina vydala. U neveřejné části byl kladen důraz na funkčnost a odolnost proti nechtěným chybám či omylům. Skládá se z převážné části z formulářů, které se starají o data vložené členy VLSI skupiny.

Abstract

This thesis deals with a design and implementation of an Internet presentation of the VLSI research group. The analytical part comprises of a selection of available technologies and standards to fulfill requirements of lucidity and usability in all common use browsers. The webpages are divided into a public and a private part. Both parts are implemented in chosen technologies (PostgreSQL a PHP). In the public part, the biggest accent was put on the safety and immunity of the application against possible attacks. In this part members of VLSI group present projects for students, news that occurred and publications that members published. In the private part an accent was put on functionality and immunity against unwanted mistakes or incompetencies. This part consists mainly of forms, that care for data inserted by members of VLSI group.

Obsah

Úvod

Cílem této bakalářské práce je vytvořit interaktivní webové stránky VLSI skupiny. Budou rozděleny na veřejnou a neveřejnou část. Ve veřejné části se návštěvník dozví o činnostech VLSI skupiny. Bude mít možnost nahlédnout do projektů skupiny, na webové stránky jednotlivých členů, zjistit si publikace, které byly skupinou vydány, či novinky, které se ve skupině odehrály. Pro návštěvníky z řad studentů bude asi nejzajímavější možnost si prohlédnout projekty, které skupina studentům nabízí, ať už jako semestrální, bakalářské nebo diplomové. Samozřejmě bude mít možnost si přečíst o projektech zadaných či běžících a případně navštívit jejich odkazy.

Do neveřejné části budou mít přístup jen členové VLSI skupiny. Každý z nich bude mít možnost pracovat s daty uloženými v databázovém systému, kde budou uloženy novinky, publikace a projekty. Členové skupiny budou moci vypisovat, editovat či mazat a v případě projektů i zadávat a ukončovat, položky v databázovém systému. K publikacím bude možné uploadovat soubory. Každý člen bude moci pracovat pouze s jím vypsanými položkami. Dále aplikace bude mít možnost nastavit takzvané administrátory, kteří budou moci pracovat i s položkami ostatních členů VLSI skupiny. Vše se bude odehrávat přes webové rozhraní, takže je kladen důraz na bezpečnost a odolnost celé aplikace.

1. Analýza potřeb a dostupných technologií

1.1. Analýza a návrh funkčnosti

1.1.1. Úvod

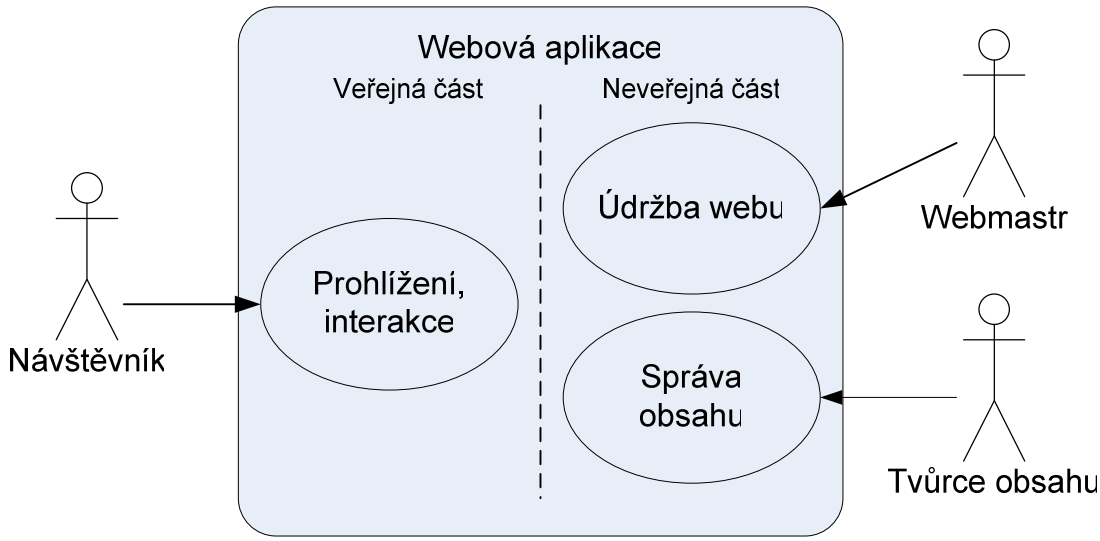
Zde bychom si měli ujasnit, jak celou aplikaci pojmeme a jaký cíl bude mít. Dále si probereme uživatelské role budoucí aplikace a hrubě si navrhneme rozložení prvků aplikace na HTML stránkách.

1.1.2. Návrh funkčnosti

Cílem aplikace je vytvořit webovou prezentaci, kde členové VLSI skupiny budou moci informovat veřejnost o své činnosti a nabízených projektech. Aplikace bude rozdělena na veřejnou a neveřejnou část. Do neveřejné části budou mít přístup pouze členové VLSI skupiny a bude zde probíhat aktualizace, správa dat a upload souborů k publikacím. Ve veřejné části bude probíhat jejich prezentace. Nyní si naznačíme uživatelské role.

Uživatelské role:

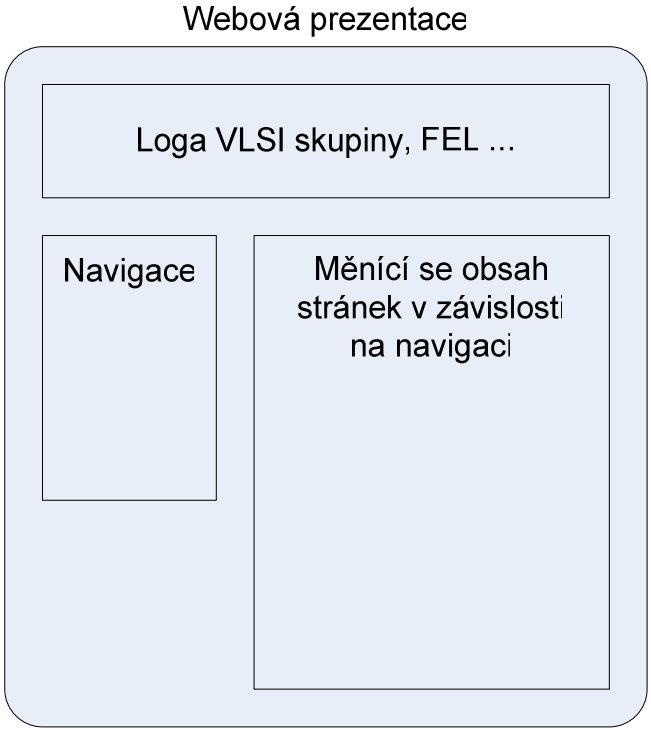
- Návštěvník – bude mít možnost prohlížet veřejnou část webové prezentace VLSI skupiny
- Tvůrci obsahu – to budou členové VLSI skupiny s možností přidávat a editovat projekty, publikace a novinky.
- Webmaster – jeden člen VLSI skupiny, který bude mít přístup ke zdrojovým kódům webové prezentace.



Obr. 1 : Uživatelské role a vztahy s aplikací

S veřejnou částí bude přicházet do styku široká veřejnost, proto by její funkčnost a rozvržení měly splňovat určité podmínky na jednoduchost a přehlednost, aby nám uživatelé neodcházeli s pocitem nepoužitelnosti tohoto webu. Proto jsem se rozhodl použít běžné rozvržení dokumentu viz. obrázek č.2, které je přehledné a uživatelům nedělá problémy. Z důvodu, že do neveřejné části budou mít přístup jen VLSI členové a bude obsahovat pouze seznam možností činností prací s daty, není rozvržení tak důležité.

Návrh rozložení prvků na HTML stránce ve veřejné části:



Obr. 2 : Rozložení prvků na HTML stránce

1.1.3. Závěr

Ujasnili jsme si, kdo vlastně bude s aplikací pracovat (členové VLSI skupiny a návštěvníci) a v jakých uživatelských rolích se budou pohybovat (návštěvník, tvůrci obsahu, webmastr). Dospěl jsem k návrhu rozložení prvků na HTML stránce tak, aby splňovaly základní požadavky na jednoduchost, přehlednost a použitelnost. S tím se pojí problém, který spočívá v tom, že každý uživatel může webovou prezentaci procházet pomocí jiného internetového prohlížeče. Bohužel se všechny prohlížeče nechovají při zobrazovaní obsahu stejně. Tímto problémem a jeho řešením se zabývám v následující části.

1.2. Aplikace na straně klienta

1.2.1. Úvod

V této části se budu zabývat problematikou prohlížečů a jejich základními rozdíly a nedostatky. Budu se snažit dospět k závěru, jehož obsahem bude výběr DTD (Document Type Declaration), aby co nejlépe pracoval s CSS (Cascadin Style Sheets) a skriptovacího jazyka pro stránky VLSI skupiny, tak, aby byly použitelné a přehledné na co největší skupině dnes používaných prohlížečů.

1.2.2. Rozdělení webových prohlížečů

Webové prohlížeče bychom mohli rozdělit na několik skupin:

a. Dle platformy – většinou chování webových prohlížečů není závislé na platformě, protože jsou většinou naprogramovány stejnou skupinou lidí nebo podle stejných pravidel, tudíž se na různých platformách chovají převážně stejně. Toto rozdělení není pro nás tak důležité.
b. Dle procentuálního zastoupení – toto rozdělení má pro nás velkou informační hodnotu a budeme se podle něho převážně řídit.

Prohlížeč	Počet návštěvníků	Procentuální zastoupení
MSIE 6.x	198838943	88 %
FireFox	14944991	7 %
MSIE 5.x	4865536	2 %
Safari	3188439	1 %
Unknown	1742429	1 %
Opera x.x	1346884	1 %
Netscape comp.	1061062	0 %
Netscape 7.x	696989	0 %
MSIE 4.x	121472	0 %
Konqueror	119806	0 %

Tab. 1 Procentuální zastoupení prohlížečů za březen 2006, zdroj: www.thecounter.com

V Tabulce č.1 vidíme, že největší procentuální zastoupení na trhu je Microsoft Internet Explorer 6.x s 88 %. Dohromady s FireFox-em a s MSIE 5.x potom zabírají okolo 97 % podílu prohlížečů na trhu. V této tabulce nejsou uvedeny všechny prohlížeče, u nichž počet návštěvníků nepřesáhl sto tisíc, měřeno na serveru zabývajícím se internetovými statistikami thecounter.com. Takovýchto serverů je na internetu více, například: OneStat, TopList atd. U každého z nich se dá najít podobná statistika. Vždy se čísla liší o několik až desítky procent. Je to dáno různými cílovými skupinami, které mají různé procentuální zastoupení prohlížečů. Hodně záleží na struktuře návštěvníků konkrétních webů. Vždy ale platí, že MSIE 6.x s FireFoxem má převážná většina návštěvníků daných webů. Výjimkou by mohly být například statistiky blogů, menších skupin lidí ,kteří mají averzi vůči Microsoftu. Statistiky takovýchto „extrémních“ případů nás však nezajímají.

Takže se v další části budeme hlavně zabývat jen prohlížeči, které mají největší zastoupení na trhu.

1.2.3. Zobrazení dat pomocí webových prohlížečů

Při zobrazování dat pomocí webových prohlížečů se můžeme dočkat velice zajímavých výsledků. Je to dáno tím, že každá skupina programátorů webového prohlížeče si může HTML tagy a CSS značky vyložit trochu jinak. Dnes je v každé HTML stránce povinné udávat definici normy, podle které se má stránka zobrazit, tzv. DTD. Přesto výrobci nejpoužívanějších prohlížečů se rozhodli vyjít vstříc autorům v období přechodu k přísným pravidlům, tudíž se nechovají podle starších definic, i když jsou v dokumentu uvedeny a tolerují programátorské chyby a zastaralé konstrukce. Díky tomuto dnešní prohlížeče pracují v tzv. různých režimech. Zpracování dokumentů nejpoužívanějšími prohlížeči bychom mohli takto rozdělit:

- STD – standardní režim
- ne-std – nestandardní režim (přechodový)
- p-STD – převážně standardní (pouze u prohlížečů na bázi Mozzilly)

V prohlížeči MSIE 6.x je nestandardním režimem způsob zobrazování dokumentů, který byl používán ještě v MSIE 5.5 pro Windows. Prohlížeč počítá chybně rozměry prvků v CSS, přehlíží lecjaké chyby, tabulky přeruší řetězec dědičnosti a skripty i CSS neinteragují dobře se stromem dokumentu. Ve standardním režimu pracuje už mnohem lépe a více se řídí specifikacemi HTML a CSS. Standardy nedodržuje úplně, ale rozdíl je to výrazný. Toto rozdělení režimů je od verze MSIE 6 pro Windows a verze MSIE 5 pro MacOS.

Prohlížeče na bázi Mozzilly mají režimy tři. V režimu standardním dodržují specifikace prakticky bezvýhradně. V režimu nestandardním se chování podobá Netscape 4.x. Oproti MSIE je tu režim třetí, tzv. převážně standardní a je to mezistupeň mezi standardním a nestandardním režimem. S co nejvyšší kompatibilitou se staršími konstrukcemi v dokumentech se současně snaží maximálně dodržovat standardy.

Podotýkám, že standardní režim v MSIE má podstatně blíže k převážně standardnímu režimu Mozzily než k jeho standardnímu.

Typ dokumentu podle DTD	NN6/MOZ	IE6/WIN	IE5/Mac
DOCTYPE chybí	ne-std	ne-std	ne-std
Před-HTML 4.0	ne-std	ne-std	ne-std
HTML 4.x Strict	STD	STD	STD
HTML 4.x Transitional bez udání URL	ne-std	ne-std	ne-std
HTML 4.0 Transitional s URL	ne-std	STD	STD
HTML 4.01 Transitional s URL	p-STD	STD	STD
XHTML 1.0 Transitional bez deklarace XML	p-STD	STD	STD
XHTML 1.0 Transitional s deklarací XML	p-STD	ne-std	STD
XHTML 1.0 Strict bez deklarace XML	STD	STD	STD
XHTML 1.0 Strict s deklarací XML	STD	ne-std	STD
ISO HTML 2000, krátká forma	STD	ne-std	ne-std
ISO HTML 2000, dlouhá forma	STD	STD	STD
ISO HTML 1999, krátká forma	STD	ne-std	ne-std
ISO HTML 1999, dlouhá forma	STD	STD	STD

Tab. 2 Aktivní režimy prohlížečů podle DTD

Legenda: ne-std = nestandardní režim; p-STD = převážně standardní režim (Mozzilla); STD = standardní režim

V Tabulce č.2 jsou uvedeny režimy, které se aktivují podle typu dokumentu DTD uvedeného v HTML stránce. Pro naše účely jsou vhodné šedě zabarvené řádky. Jsou to ty, kdy se většina dnešních prohlížečů chová dle standardů. Z uvedené nabídky jsem zvolil : **XHTML 1.0 Transitional bez deklarace XML**. Tento uvedený typ zároveň umožňuje správci webu používání starších tagů a konstrukcí, na které je zvyklý.

1.2.4. Skriptovací jazyky na straně klienta

Na straně klienta budeme chtít kontrolovat formuláře zadávané členy VLSI skupiny, aby zbytečně neprobíhala komunikace mezi serverem a klientem, pokud bude formulář zadán chybně. K tomu budeme potřebovat krátký program napsaný v jazyce, který poběží na straně klienta. Zde je přehled nejpoužívanějších jazyků, které běží na straně klienta:

- Java (například formou apletu)
- Flash aplikace
- VBScript (Visual Basic Scripting)
- JavaScript

Java a Flash aplikace budou pro náš záměr zbytečný luxus. U těchto jazyků se klientovi posílá už zkompilovaný kód a na straně klienta je prohlížeč interpretuje pomocí nainstalovaných součástí. Navíc tyto jazyky může mít uživatel vypnuté, či nenainstalované.

POZN: JavaScript není Java. V HTML stránce, případně v přilinkovaném souboru je JavaScript v nezkompilované podobě. Klient si jej může zkopírovat a editovat. To nám vadit nebude. Budeme ho používat jen na všeobecně známé kontroly formulářů.

Zbývá tedy VBScript a JavaScript. Oba jazyky se interpretují na straně prohlížeče a lze je implementovat přímo do HTML pomocí tagů. Jsou to velmi podobné jazyky co do možností a cílů autorů. Liší se především syntaxí a drobnými rozdíly v možnostech jazyků. Hlavní rozdíl však plyne z historie obou jazyků. Společnost Netscape pro podporu svého prohlížeče vytvořila jazyk JavaScript. Microsoft reagoval tím, že ho implementoval do svého prohlížeče a zároveň si vymyslel svůj vlastní: Visual Basic Scripting, jenže společnost Netscape ho neimplementovala. A tak je to do dnes. VBScript je výhradou MSIE a tudíž není funkční ve všech ostatních prohlížečích, z čehož jednoznačně plyne, že použijeme JavaScript.

I JavaScript lze bohužel na straně klienta deaktivovat, či jednoduše obejít, takže se neobejdeme bez úplné kontroly na straně serveru.

1.2.5. Závěr

Rozborem trhu jsem dospěl k závěru, že pro danou aplikaci použijeme DTD : **XHTML 1.0 Transitional bez deklarace XML** z důvodů normovaného chování většiny dnešních prohlížečů a zároveň možnosti používání starších konstrukcí v HTML kódu. Jako jediný použitelný a vhodný skriptovací jazyk nám z rozboru vyplynul **JavaScript**, který nám umožní částečně omezit komunikaci mezi serverem a klientem při špatném vyplnění formuláře.

1.3. Softwarové vybavení serveru

1.3.1. Úvod

Budu se zde zabývat analýzou a porovnáním softwarového vybavení serverů, které se jsou u nás na trhu. V závěru bychom měli dospět k seznamu softwaru, který bude potřebný pro běh webové aplikace na straně serveru.

1.3.2. Skriptovací jazyky

Pro běh naší aplikace na straně serveru budeme potřebovat nějaký skriptovací jazyk, který bude zpracovávat požadavky a vytvářet na ně odpovídající odpověď. Na našem trhu takovýchto prostředků je několik, zde uvádím jen ty nejzákladnější a nejznámější :

- Perl
- Python
- Cold Fusion
- ASP
- PHP

Perl a Python jsou velice silné nástroje pro práci na straně serveru. Oba jazyky toho umí velice dost. Perl je dnes dostupný v každé systému typu Unix/Linux. Python zatím takovéto rozšíření nemá, ale v poslední době začíná částečně Perl vytlačovat. Pro nás tyto jazyky nejsou tak zajímavé, protože ani jeden z nich nebyl přímo navržený na skriptování pro web. Z tohoto důvodu jsem je zavrhl.

Cold Fusion má velice dobré zpracování chyb, databázovou abstrakci a parsování dat. Bohužel se o něm tvrdí, že je pomalejší a méně stabilní něž například PHP. Cold Fusion je velice jednouchý jazyk pro začátečníky a bylo navrženo s ohledem na „neprogramátory“ oproti PHP a proto pro nás není vhodným kandidátem.

Zbývají nám tedy dva: PHP vs. ASP. ASP není ve skutečnosti jazyk jako takový, je to zkratka Active Server Pages. Jazyky nyní používanými k programování ASP jsou Visual Basic Script a Jscript. Největší nevýhodou ASP je to, že se jedná o systém, který je nativně používán pouze na serveru Microsoft Internet Information Server. To silně omezuje jeho dostupnost. ASP je mnohem více objektově orientováno než PHP, i když PHP ve verzi 5 udělalo velký skok k dobré práci s objekty. V následující tabulce si můžeme prohlédnout vlastnosti a porovnání obou jazyků.

	PHP 4	PHP 5	ASP.NET
Cena Softwaru	zdarma	zdarma	zdarma
Cena platformy	zdarma	zdarma	$$
Rychlost	vysoká	vysoká	slabší
Efektivnost	vysoká	vysoká	slabší
Bezpečnost	vysoká	vysoká	vysoká
Nezávislost na platformě	vysoká	vysoká	slabá
Dostupnost zdrojových kódů	ano	ano	ne
Výjimky	ne	ano	ano
OOP	slabé	vysoké	vysoké

Tab. 3 Porovnání ASP a PHP, zdroj: http://www.oracle.com/technology/pub/articles/hull_asp.html

Položka ceny softwaru není tak jednoznačná, i u PHP se můžete dočkat peněžních výdajů, například při investici do optimalizačního Zend enginu. ASP vám umožňuje používat různé programovací jazyky a je silně objektově orientovaný. To však vede k tomu, že pro zpracování stejné úlohy se musí v ASP zpracovat větší množství kódu a tím nám dává ASP o něco horší výsledky v rychlosti a efektivnosti než PHP.

ASP běží na Microsoft Internet Information Server a začíná se dostávat i na Apache, který může běžet na různých platformách. PHP bylo od začátku vyvinuto přímo pro práci na systému Apache a to mu dává větší nezávislost na platformě.

Na základě těchto informací jsem dospěl k názoru, že pro náš účel bude lepší použít skriptovací jazyk PHP. Nahrává nám k tomu i skutečnost, že stránky budou pravděpodobně provozovány na serveru service, na kterém právě běží Apache a PHP je tam už implementováno.

1.3.3. Úložiště dat

Data, která budou zadávat VLSI členové v neveřejné části, se budou muset někde skladovat. Nejedná se o žádnou složitou strukturu dat, takže v podstatě máme dvě možnosti, jak data uchovávat:

- Souborový systém serveru
- Databáze

Souborový systém serveru:

Prvotní nápad byl, protože mezi daty nejsou žádné složité vztahy, obejít se bez databáze a data skladovat v souborovém systému serveru. Tímto směrem se ubíraly i mé prvotní myšlenky. Pro zadání potřebujeme skladovat tyto měnící se položky:

- **Projekty**
  - Nabízené
  - Běžící
  - Ukončené
- **Novinky**
- **Publikace** a soubory k nim nahrané členy VLSI skupiny

Myšlenka tedy byla taková, že pro každou z těchto položek budeme mít samostatný soubor, ve kterém bude v nějaké podobě uložena tabulka s danými položkami. Pro soubory, které VLSI členové nahrají na server, jsem vyhradil složku. Cesta a název souboru by se ukládaly jako položka do tabulky publikace.

Dolování a ukládání dat ze souborů v PHP:

Jazyk PHP je relativně silný v práci se soubory a řetězci. Nabízí nám následující funkce, které jsou pro daný účel jak stvořené:

- řetězec ***implode*** (separator, array)
- řetězec ***explode*** (separator, řetězec)

Jedná se o opačné funkce. ***Explode*** vrací pole řetězců, z nichž každý je částí argumentu vstupního řetězce, vzniklý jeho rozdělením na hranicích tvořených zadaným řetězcem separator. Naopak funkce ***implode*** vrací řetězec obsahující řetězcovou reprezentaci všech prvků pole v původním pořadí s řetězcem separator mezi každými dvěma prvky. S těmito funkcemi je práce po načtení souboru snadná. Obsah souboru rozdělí na jednotlivé položky a dále se s nimi může pracovat jednotlivě.

V případě, že bychom chtěli ze souboru jen číst, tak by nám tenhle systém fungoval. My ale budeme chtít do souboru i zapisovat, tudíž se budeme muset postarat o to, aby v případě práce několika členů VLSI skupiny najednou se nám nesmazaly či neuložily některé položky. Budeme potřebovat vytvořit nějaké zámky, které nám zajistí, že s daným souborem bude pracovat pouze jeden člověk.

Možnosti jak vytvořit zámek pro soubory v PHP:

- fce ***mkdir*** kontrola existence adresáře
- fce ***shmop_****xxx* pro práci se sdílenou pamětí
- fce ***sem_****xxx* semafor pro sdílenou paměť
- fce ***flock*** zamykání souboru

Kontrola existence adresáře:

Tento systém je v širokém internetu relativně hojně používán. Spočívá v tom, že předtím než otevřeme soubor, tak se funkcí ***mkdir*** pokusíme vytvořit adresář stejného či pozměněného jména a v případě úspěšného vytvoření, považujeme zámek za zamčený. Funkce ***mkdir*** v případě existence již daného adresáře vrací false a v tomto případě se chováme tak, že zámek zamkl jiný uživatel. Dále můžeme čekat, než daný uživatel skončí práci se souborem a adresář smaže. Tento systém je funkční pouze při malém provozu na souborech, co do nich zapisujeme, a může selhat, protože funkce ***mkdir*** není atomická. Z tohoto důvodu by se vlastně jednalo o velmi nekvalitní typ zámku a proto jsem ho zamítl hned.

Funkce shmop_xxx :

Jazyk PHP nám nabízí funkce ***shmop_***xxx, které pracují s jednoduchou reprezentací sdílené paměti. Dá se použít i pro výměnu dat do programů napsaných v jiném jazyce např. C. Funkce ***shmop***_xxx má při vytváření bloku sdílené paměti na výběr z několika přepínačů. Pomocí přepínače „a“ otevřeme existující blok sdílené paměti. Pomocí přepínače „c“ vytvoříme nový segment sdílené paměti. V některých dokumentacích jsou uvedeny ještě další, pro nás zajímavý přepínač „n“, který, když použijeme, tak funkce v případě že blok paměti neexistuje ho vytvoří a v případě, že existuje tak vrací false. Toho by šlo využít stejně jako u předchozího případu s funkcí ***mkdir***.

Bohužel přepínač „n“ není uveden ve všech dokumentacích, a po mém testování na Windows se funkce nechovala tak jak má. To je možná důvod, proč není uveden všude, protože nemá na všech platformách stejnou implementaci. Ani jsem se nikde nic nedočetl o atomicitě těchto funkcí, takže jsem je z těchto důvodů také zavrhl.

Semafory v PHP pro sdílenou paměť – funkce sem_xxx

Po vytvoření segmentu sdílené paměti pomocí funkcí typu ***shm_***xxx lze přístup řídit pomocí klasických semaforů , které v PHP najdeme za funkcemi ***sem_***xxx. U nich je atomicita v podstatě zaručena, protože byly k tomuto účelu přímo vyvinuty. Naproti tomu je jejich používání pro náš případ až zbytečně složité. Navíc práce se sdílenou pamětí pomocí

funkcí ***shm_***xxx není v klasické verzi PHP implementována a musí se do systému doinstalovat. Dále tyto funkce nefungují na všech platformách a proto jsem od nich také upustil.

Funkce *flock* :

Je vyvinuta přímo pro naše potřeby zamykání souborů. Je to tzv. „portable“ způsob zamykání celých souborů na základě jednotného „advisory“ přístupu, což znamená, že všechny přistupující programy musí používat stejný způsob zamykání, jinak by to nefungovalo. Bohužel je funkce ***flock*** na většině systémech implementována na úrovní procesů. Takže při použití multithreadového serverového API nelze spoléhat na ochranu souborů proti jiným skriptům běžících v paralelních vláknech. V novějších verzích bude implementace možná lepší, ale na to se nemůžeme spoléhat. Náš projekt by měl běžet na serveru, který multithreadový není, ale do blízké budoucnosti není zaručena jeho výměna a z důvodů dlouhodobější použitelnosti tohoto projektu jsem tuto možnost také zavrhl.

Z rozboru možností PHP při práci se soubory jsem nenašel pro naše účely vhodný a silný nástroj, takže jsme se rozhodl jako úložiště dat použít databázi.

Databázové systémy:

Zde jsou vedeny nejdostupnější databázové systémy z hlediska podpory a ceny:

- MySQL
- PostgreSQL

Oba tyto systémy jsou stabilní, zdarma a v dnešní době je lze provozovat na většině platformách. Pro MySQL mluví to, že je ve většině kategoriích rychlejší než PostgreSQL, nicméně rychlost je zde vykompenzována tím, co databáze PostgreSQL umí. Dále MySQL méně dodržuje standardy SQL 92, 95. MySQL se snaží svými možnostmi dohánět PostgreSQL, ale nejde jí to tak rychle, jak by chtěla. PostgreSQL prošla delším vývojem a i to se odrazilo v její bezpečnosti.

Jediná nevýhoda databázového systému PostgreSQL je v její rychlosti, a proto je pro naší aplikaci vhodnější. Skutečnost, že na serveru, na kterém by měla být aplikace provozována, je PostgreSQL již implementována, nás jen utvrdil ve zvolení tohoto databázového systému.

1.3.4. Bezpečnost a odolnost

Vytvořená aplikace musí být dostatečně bezpečná a odolná proti možným útokům z širokého internetu. V případě neveřejné části by se dalo mluvit pouze o odolnosti proti

nechtěným, omylem zadaným hodnotám, protože se jedná o malou skupinu lidí, kteří nemají zájem na systém útočit. Na to se však nemůžeme spoléhat. Proto zde rozdělím otázku bezpečnosti na dvě části:

Veřejná část:

Nejbezpečnější aplikace by byla taková, že od uživatelů neobdržuje žádná data a tudíž ji nemají jak napadnout. Takovéto stránky se dají napsat jako statické HTML, ale v našem případě, kdy částí bude hodně a často se mění, by to bylo neúnosné na aktualizaci. Proto se omezíme na minimum přijímaných dat. Všechny hodnoty se budou přenášet metodou GET (tzv. budou vidět v odkazu na stránku). Možný útok na takovouto aplikaci spočívá v tom, že uživatel změní hodnotu proměnné v odkazu. To může být nebezpečné například tehdy, pokud hodnota proměnné představuje jméno stránky a zároveň vkládaného souboru. Útočník může změnit hodnotu na url adresu a docílit tím načtení serverem svého skriptu, ve kterém si poté může dělat co chce.

Ochrana Veřejné části bude spočívat v kontrole všech proměnných předávaných metodou GET na přípustné hodnoty. Zvláště pak proměnné, ve které se předává jméno na odkazovanou stránku. Nesmíme zapomenout ani na výpis hodnot z databáze tak, aby nám nerozhodily námi navrženou strukturu webu a jeho validitu.

Neveřejná část:

Ochrana neveřejné části bude spočívat také v kontrole všech předávaných proměnných metodou GET. Na rozdíl od veřejné části se zde budou používat i formuláře. V některých formulářích se bude pracovat i se soubory. Takováto práce vyžaduje metodu přenosu POST. Proto budeme při reakci na formulář kontrolovat i vše předávané metodou POST.

Další problém může nastat při vkládání dat do databáze.

- Opakované vkládání dat při mačkání Reloadu ,či pohybu šipkami vzad a dopředu v prohlížeči. Tomu lze zabránit přesměrováním na jinou stránku po vložení dat pomocí funkce ***header*** v PHP.
- Opakované odesíláni formuláře: např. vícenásobné zmáčknutí tlačítka odeslat při pomalé odezvě serveru. Zjistíme to pomocí ***SESSION*** tak, že každý uživatel bude moci mít otevřen od každého typu jen jeden formulář a po jeho odeslání se vloží záznam do DB jen jednou. Pro vložení dalšího záznamu musí uživatel formulář znovu načíst.
- Útok na databázi, či vkládáni nevhodných řetězců. V databázi PostgreSQL se řetězce dávají mezi jednoduché uvozovky, pokud by se ve vkládaném řetězci

tento znak vyskytoval, tak se daný řetězec ukončí a to, co je za ním, se vyhodnotí jako příkaz. To může způsobit značné problémy a dovolit útočníkovi dělat si s databází, co chce. Pomoc je jednoduchá a spočívá v escapování nebezpečných znaků. V PHP máme takovéto funkce k dispozici.

V posledním případě může útočník chtít se dostat do neveřejné části, i když na to nemá oprávnění. Kontrola tohoto nám odpadá, protože na serveru, na kterém se bude aplikace provozovat, je možnost použít SSL. Takže neveřejná část se dá do části serveru, kde se vyžaduje přihlášení uživatele přes SSL a ochranu nám tímto poskytuje sever Apache bez našeho zásahu či nastavení.

1.3.5. Závěr

Po analýze dostupných technologií jsem vybral skriptovací jazyk PHP a jako úložiště dat databázový systém PostgreSQL. Oba tyto systémy jsou dostatečně výkonné pro naší aplikaci a splňují požadavky na bezpečnost. Navíc jsou už na serveru, na kterém bude naše aplikace běžet, už implementovány, takže odpadá jejich instalace a nastavování.

2. Návrh Řešení

Celá aplikace je rozdělena na veřejnou a neveřejnou část. Jako celek bude aplikace dávat možnost prezentace VLSI skupiny na internetu. Data budou uchovávána ve vybraném databázovém systému. Velký důraz je kladen na bezpečnost aplikace.

2.1. Veřejná část aplikace

Veřejná část bude dostupná všem na internetu, proto je zde důležitá otázka bezpečnosti. Aplikace z veřejné části nebude tedy nic ukládat do databáze. Z databáze bude jen načítat data podle předem připravených dotazů, aby nemohlo dojít k narušení databáze. Všechna interakce mezi webem a uživatelem bude přísně kontrolována. Data se budou předávat metodou GET, v podstatě se jedná jen o výměnu, požadované stránky nebo o listování na dané stránce. Grafický návrh bude muset splňovat požadavky na přehlednost jednoduchost a intuitivnost.

2.2. Neveřejná část aplikace

Do neveřejné části budou mít přístup pouze členové VLSI skupiny. Bude jim umožňovat vkládat a editovat data v databázovém systému. Bezpečnost je dána tím, že tato sekce bude v části serveru, kde je vyžadováno přihlášení přes SSL. Ze strany členů VLSI skupiny se nepředpokládá žádný útok, přesto jsou ale v aplikaci kontrolována všechna předávaná data mezi členy a serverem. Soubory uploadované k publikacím budou ukládány do nastaveného adresáře a v databázi budou uchovávány jen jejich názvy. Ovládání neveřejné sekce musí být jednouché, přehledné a intuitivní. Grafická úprava v této části není důležitá.

3. Řešení

V této kapitole se budu zabývat řešením které, jsem zvolil a jeho implementací. Nejdříve rozdělení adresářové struktury. Najdeme zde tyto složky :

- Download – v této složce je index.php, který by měl být přesunut do stejnojmenné složky na serveru service.
- Internal – V této složce se nachází neveřejná část. Měla by být přesunuta do části serveru service, kde je vyžadováno přihlašování, tudíž někam k interním stránkám VLSI skupiny
- Service_PHP – zde jsou uloženy skripty pro připojování k DB
- Zdroj – do této složky se ukládají uploadované soubory k publikacím. Musí mít vhodně nastavené práva, aby do ní mohl zapisovat skript z interní sekce.

A samozřejmě se v kořeni nachází několik PHP skriptů, které jsou veřejné a starají se o zobrazování obsahu.

3.1. HTML a CSS

Kaskádové styly jsou uloženy v souboru *cssphp.css*. Je kladen důraz na to, aby celé stránky byly plně validní podle DTD, které jsem zvolil v analýze:

XHTML 1.0 Transitional bez deklarace XML

Ani toto striktní dodržování validity nám však nezaručí stejné zobrazování na všech prohlížečích. Pokud bereme v úvahu jen ty, které mají na trhu ještě větší procento zastoupení, tak dělá problémy hlavně Internet Explorer verze 5.x, který má problémy hlavně se špatným počítáním box-modelu prvků a i s okraji (padding, margin) řádkových prvků, které zobrazuje bez okrajů.

3.1.1. Navigační menu

Je to asi nejdůležitější část, která musí fungovat vždy a všude, proto je v HTML udělána pomocí seznamů a odkazů, se kterými si poradí všechny dnes používané prohlížeče. Celé menu je uzavřené do tagu *div,* aby se s ním dalo pracovat jako s celkem. Zde je ukázka:

```
<div id="navigation">
<ul>
<li><a href="index.php?page=main">Home</a></li>
<li><a href="index.php?page=novinky">Novinky</a></li>
<li><a href="index.php?page=people">Lidé</a></li>
<li><a href="index.php?page=research">Výzkum</a></li>
…
</ul>
</div>
```

V souboru *cssphp.css* si pak můžete prohlédnout, jak je s menu pracováno na úrovni kaskádových stylů. Pro celé menu není nastavena velikost písma, to je uděláno jen pomocí relativních hodnot *em* (velikost písma právě používaného prvku) v CSS. Je to proto, aby se každému uživateli zobrazovala navigace dle jeho potřeb a nastavení prohlížeče, aby nedocházelo k situacím, kdy nemůže někdo menu přečíst díky špatnému zraku či jiným problémům. Menu se plně přizpůsobí nastavení velikosti písma v prohlížeči, můžete si to vyzkoušet například v IE, tak že dáte záložku zobrazit a vyberete položku velikost písma.

Tam kde nejsou vlastnosti *padding* a *margin* nastaveny, tak jsou nulovány, protože prohlížeče si bez udání jejich velikosti, berou svojí přednastavenou hodnotu a u každého typu prohlížeče je jiná. Proto bychom se bez tohoto opatření dočkali různého rozložení prvků na stránce. U tagu *li* pomocí vlastnosti *list-style* se zbavujeme zobrazování odrážek u seznamu, protože se nám do navigace nehodí. Je zde nastavena i velikost fontu pomocí vlastnosti *font-size* na hodnotu *1em*. Je to sice výchozí nastavení všech prohlížečů, ale Internet Explorer při nezadané hodnotě špatně počítá velikost fontu, ale pokud mu ji předstrčíme, tak ji spočítá správně.

Dále odkazům v seznamu nastavíme šířku a necháváme je zobrazit blokově. Je to proto, aby se na navigaci dalo kliknout v celé její šířce a ne jenom přímo nad odkazem. Při najetí nad odkaz používáme změnu barev pozadí a okrajů tak, aby aktivní odkaz byl rozlišen od ostatních.

Navigační menu je plně funkční ve všech prohlížečích. Korektně se zobrazuje v IE 6.x a Mozzille. Starší verze Internet Exploreru špatně počítají box-model, a proto dochází k estetickým chybám při jeho zobrazení, které nemají vliv na funkčnost. Vše zobrazuje správně, až do chvíle najetí myší nad navigační menu. Plocha, na kterou se dá kliknout, je mírně ořízнuta zprava, to však nic nemění na jeho plné použitelnosti a funkčnosti. Jelikož je ořízнuta pouze plocha padding, která při přejetí plochy není aktivní, tak je tento jev nepříjemný pouze esteticky.

Řešení tohoto problému by spočívalo v použití různých souborů CSS stylů v závislosti na prohlížeči. Když by byl detekován tento prohlížeč, tak by byl do HTML souboru přilinkován jiný CSS soubor. I toto řešení však není stoprocentní, protože prohlížeče nemusí odesílat svoji identifikaci a tudíž pak nelze určit, který CSS soubor by měl být přilinkován. Dále by přibyla povinnost při případné grafické změně upravovat kód na více místech.

3.1.2. Zbytek dokumentu

Celý dokument je rozvržen tak, jak bylo nastíněno v analýze na obrázku č.2. Pomocí tagů *div* jsem obsah rozdělil na navigační menu a měnící se obsah. Nad těmito položkami se ještě nacházejí loga.

Snažil jsem se u log dosáhnout takového zobrazení, aby byli přehledné a souměrné. To je dosaženo absolutním pozicováním od okrajů zobrazované plochy. Dále jsou u navigačního Menu odkazy na jazykové verze a na interní stránky. U odkazu na interní stránky jsem nastavil šedou barvu, protože tento odkaz je určen pouze členům VLSI skupiny a pro ostatní není důležitý.

Odkazy na jiné jazykové verze jsou pomocí obrázků. U těch jsem zrušil vlastnost border, protože pokud jsou ohraničeny modrým pruhem jako odkazy, tak vypadají dosti nevzhledně a nezapadají do grafické koncepce stránek. Na první pohled by sice nemuselo být jasné, že se jedná o odkazy, ale ve světě jsou takovéto vlaječky využívány převážně právě jako přepínače jazykové verze a uživatelé jsou s nimi již seznámeni, takže by jim jejich používání nemělo dělat problémy.

Jak je známo, tak pro formátování by se dnes už neměly používat tabulky, ale CSS. To je v mém návrhu dodrženo. Jediná tabulka, která je zde použita, je při zobrazování uživatelů a jejich webů. Každý prohlížeč si tabulky zobrazuje opět po svém. Stejného vzhledu je zde dosaženo tím, že je použito více principů. Pro starší prohlížeče je k dispozici u tagu *td* starý atribut width, který určuje procentuální velikost buňky vzhledem k velikosti tabulky. Pro prohlížeče, které se řídí standardy, je v CSS uvedena velikost, a blokové zobrazení. Tím je dosaženo, že prohlížeče vezmou na vědomí i velikosti výplní, takže tabulka je přehledná a čitelná.

Celý dokument má nastaven jednoduchý a jednotný vzhled tak, aby prohlížení dokumentů bylo přehledné a grafická prezentace nemlátila do očí. Pro zpřehlednění záznamů v seznamech jsem vybral dvě barvy, které se střídají a jsou snadno rozlišitelné, ale nekonfrontují mezi sebou, aby se usnadnila orientace v delších záznamech.

Na následujícím obrázku si můžete prohlédnout ukázku rozložení prvků a grafického výstupu prezentace:

Obr. 3 : Ukázka stránek VLSI

Aplikace je plně validní bez chyb a warningů, a to jak generovaný HTML kód, tak kaskádové styly. HTML kód byl kontrolován validátory:

- http://validator.w3.org/ - online validace HTML kódu
- Rozšířením *Html Validator 0.7.9* pro prohlížeč FireFox s úrovní přístupu nastavenou na *normální*.

Kaskádové styly byly kontrolovány:

- http://jigsaw.w3.org/css-validator/

3.2. Databázový systém

V analytické části jsme se dobrali k tomu, že jako úložiště dat budeme používat databázi PostgreSQL. Po konzultaci s vedoucím projektu jsme se dobrali k seznamu položek, který je potřeba v databázi skladovat:

- **Studentské projekty**
  - Nabízené
  - Běžící
  - Ukončené
- **Novinky**
- **Publikace**

3.2.1. Struktura tabulek

Publikace a Novinky budou v samostatných tabulkách. Je u nich potřeba uchovávat následující informace:

Novinky:

- **id** – primární klíč
- **from_date** – datum, kdy byla novinka zadána
- **to_date** – datum, kdy vyprší platnost
- **název**
- **body** – text novinky
- **zadal** – jméno zadavatele

Publikace:

- **id** – primární klíč
- **autoři** – jména autorů publikace
- **popis**
- **rok** – rok vydání publikace
- **zdroje** – jména souborů uploadovaných na server
- **zadal** – jméno zadavatele

Studentské projekty by se daly skladovat v jedné tabulce. Ale já jsem se rozhodl pro dvě. Ukončené projekty budou mít svoji vlastní tabulku. V ní se postupem času budou hromadit ukončené projekty. O nabízené a běžící projekty, z hlediska studentů, je větší zájem, a tak tímto způsobem jsem se snažil ulehčit databázi. Při výpisu těchto projektů se nebude pracovat

s ukončenými projekty, které budou v jiné tabulce. Struktura tabulek bude shodná až na malé výjimky. U projektů je třeba uchovávat následující informace:

Projekty:

- prjid – primární klíč
- název
- nazeveng – anglický název
- popis – u ukončených projektů se tato položka jmenuje **anotace**
- rozsah – rozsah projektu, SP, BP, DP
- literatura – doporučená literatura k projektu
- zadavatel – jméno zadavatele
- řešitel – řešitel projektu
- odkaz – odkaz na stránku projektu
- zadano – booleovská hodnota, zda je projekt zadán (není u ukončených projektů)
- typ – číselná reprezentace typu projektu :
    - 1- softwarový
    - 2- hardwarový
    - 3- kombinace software / hardware
    - 4- rešeršní

Dále je potřeba někde udržovat informaci o členech VLSI skupiny. Na serveru service už databáze členů existuje a ze zabezpečené části serveru, kam mají členové VLSI stránek přístup, je dostupná. Tudíž jsem se rozhodl, že není potřeba dané tabulky udržovat v naší databázi a tím způsobovat redundanci dat. Sice se nám tím komplikuje trochu aplikační logika stránek, jelikož budeme muset nejdříve sáhnout do jedné databáze pro členy VLSI, případně pro jméno přihlášeného člena, a pak následovně sáhnout do naší databáze pro požadovaná data.

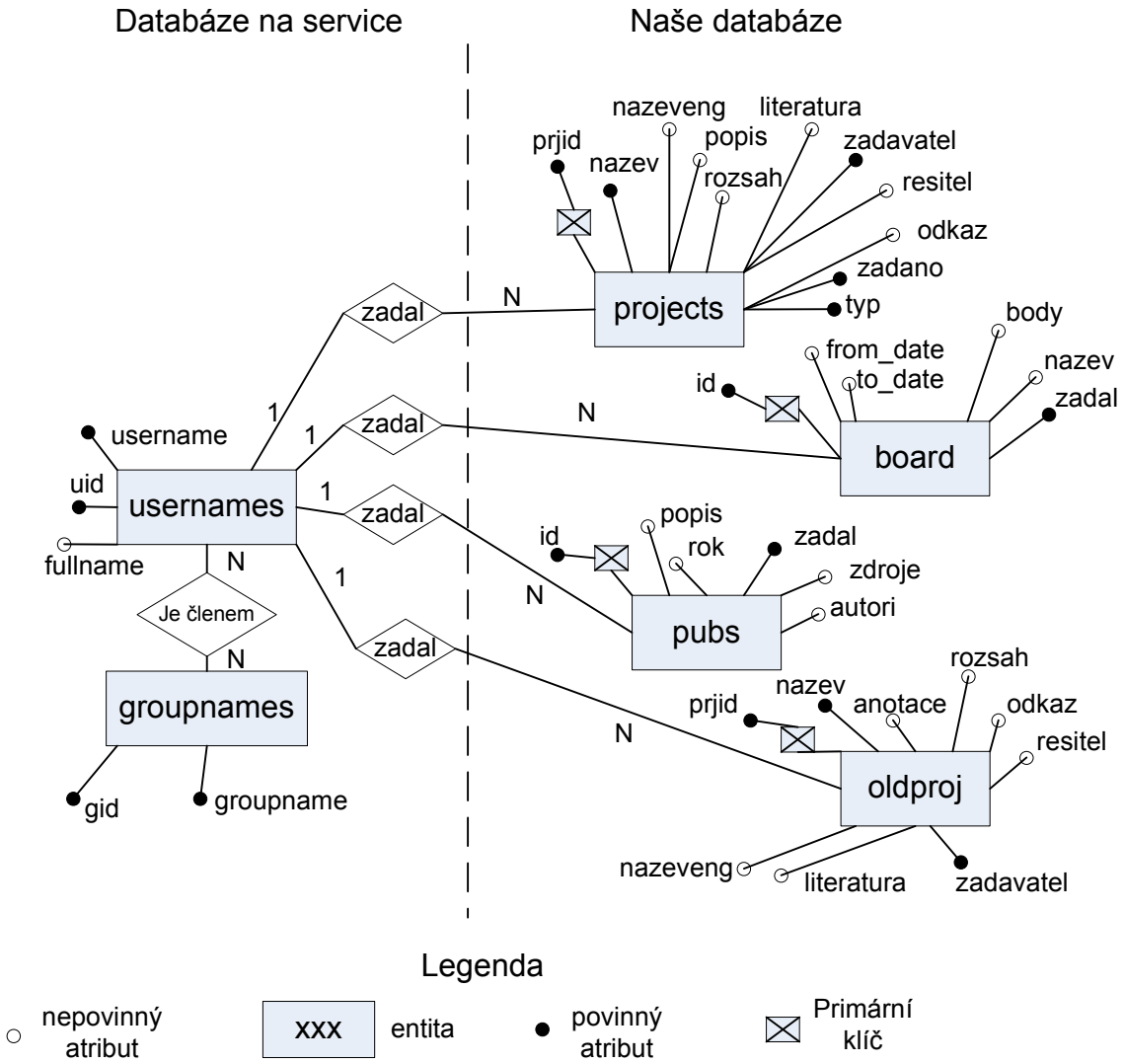
Do sloupců **zadavatel** u projektů a **zadal** u publikací a novinek tedy budeme ukládat login daného člena skupiny. Šlo by tam ukládat i primární klíč, ale tím bychom ztratili přehlednost naší databáze. U loginů je zajištěna jejich unikátnost, takže pro tento účel se dají použít.

3.2.2. ER model

Názvy entit:

- **projects** – projekty nabízené a běžící
- **oldproj** – staré projekty

- **board** – novinky
- **pubs** - publikace



Obr. 4 : ER model databáze

Na obrázku č.4 si můžete prohlédnout ER model navrhnuté databáze. Část vlevo, která již běží na serveru service a není součástí naší aplikace, je co nejvíce zjednodušená. Vztahy mezi těmito dvěma databázemi pro nás znamenají proces, při kterém se postupně připojíme k jedné databázi a potom k druhé. Tento relativně nestandardní postup je proto, abychom nemuseli v naší databázi skladovat informace o členech a tím mezi databázemi vytvořit redundanci dat.

3.2.3. Jiná možná řešení

Nabízela se možnost vytvořit v databázi, oproti uvedenému návrhu, navíc tabulku se členy VLSI skupiny. Tímto způsobem by se nám zredukovalo připojování do dvou databází na jedno, ale na server bychom tím zanesli redundanci. Informace o členech VLSI skupiny by byly skladovány na dvou místech. Přibyla by starost o aktualizaci přidané tabulky členů. Dalo by se i ulevit databázi, protože by v ostatních tabulkách mohl být atribut zadavatel (resp. zadal) číselné hodnoty, a tudíž by vyhledávání probíhalo rychleji.

3.2.4. Ukázka použitého SQL

Zde je ukázka použitého SQL příkazu pro vytvoření tabulky **projects** :

```
CREATE TABLE projects
(
  prjid serial NOT NULL,
  nazev varchar(320) NOT NULL,
  nazeveng varchar(320),
  popis text,
  rozsah varchar(40),
  literatura varchar(320),
  zadavatel varchar(8) NOT NULL,
  resitel varchar(40),
  odkaz varchar(160),
  zadano bool NOT NULL DEFAULT false,
  typ int2 NOT NULL DEFAULT 1,
   PRIMARY KEY (prjid)
);
```

V této ukázce by mělo být vše jasné. Ve většině databázích se tabulky vytvářejí stejně. Jediná zvláštnost na tomto příkladě by mohla být u primárního klíče typ *serial*. Je to vlastně vytvoření vlastní sekvence pro danou tabulku, která se používá pro dosazování hodnot do primárního klíče (něco jako auto_increment u MySQL).

Příkazy pro ostatní tabulky jsou podobné, a proto je zde nebudu uvádět. Můžete je najít v příloze nebo na přiloženém CD v souboru: tabulky.sql

3.3. PHP – Aplikační logika

3.3.1. Práce s databází

Základní soubory pro práci s databází mi byli dány k dispozici. Jednalo se o několik souborů, které zajišťují autentizaci uživatele na serveru service. Využil jsem z nich několik souborů.

U definice objektu, který se stará o zpracování chyb nastalých při komunikaci s databází, jsem označil část skriptu, který zapisuje nastalé chyby do souboru ,jako komentář. V případě potřeby případného doladění, se může tato změna vrátit zpět. U objektu pro práci s databází byla funkčnost nezměněna, protože na serveru service úspěšně běží. Změnil jsem tam několik drobností pro vlastní potřebu, a to následující věci:

- Zrušil jsem vytváření objektu chyby při chybné komunikaci s databází. Je to proto, že připojovat k databázi se bude i ve veřejné části a tímto zajistíme, že se uživatel při případné nefunkčnosti databáze, či jiného problému, nedozví nic o struktuře naší databáze z chybových hlášek. Vytváření objektu chyby je označeno jako komentář, takže se dá pro případné ladění snadno použít.
- Zrušil jsem výpis dotazu při jeho špatné formulaci, či chybě DB, a to ze stejných důvodů jako výše, aby se uživatel nedozvěděl nic o struktuře DB.

Seznam členů VLSI skupiny načítáme z databáze následujícím dotazem.

```
SELECT u1.username,u1.fullname
FROM usernames u1
 WHERE u1.uid IN(        SELECT g1.uid
                         FROM groups g1
                         WHERE g1.gid=(          SELECT g2.gid
                                                 FROM groupnames g2
                                                 WHERE g2.groupname='VLSI'));
```

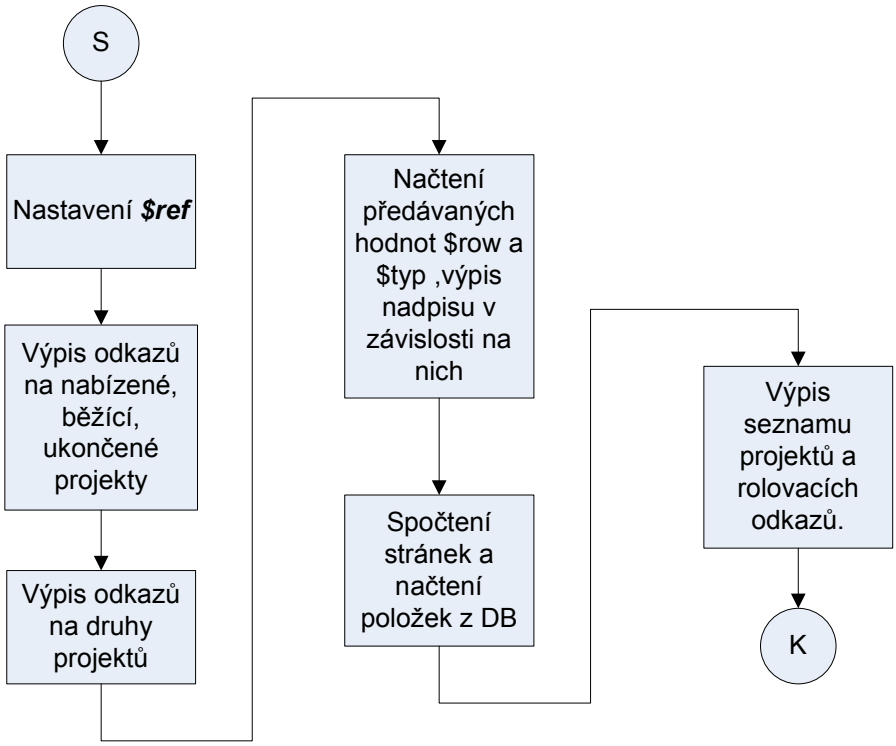
Dotaz je tvořen poddotazy, protože ve většině případů jsou rychlejší než náročné spojování tabulek s mnoha řádky. V nejnižším dotazu v této struktuře zjišťujeme primární klíč jména skupiny. Ve druhém dotazu vybereme klíče uživatelů, kteří do dané skupiny patří. A ve třetím nejvrchnějším dotazu, se ptáme na jména těch, kterým dané klíče patří. Výsledek dotazu je uložen do asociativního pole ***$jmena[]***, pro pozdější použití ve skriptech.

V případě, že by nám na stránkách nefungovala čeština, tak v souboru **dbconn.php** je část kódu, který zajišťuje, aby databáze s námi komunikovala v naší znakové sadě. Tato část kódu je jako komentář, protože, aby Postgres mohl převádět znakovou sadu, musí mít danou databázi vytvořenou s rozšířenou znakovou sadou, např. UTF-8. V případě, že dotaz

nefunguje, tak má Postgres databázi vytvořenou v ASCI podle nastavení daného počítače, což nám nevadí a převod nepotřebujeme, protože na daném počítači je čeština. Při uvádění projektu do provozu zjistíme či vyzkoušíme nastavení serveru a případně zrušíme komentářové značky u daného kódu.

3.3.2. Projekty – nabízené, běžící, ukončené

Kód těchto tří skriptů je velice podobný, a proto to shrnu do jednoho popisu, který si můžete prohlídnout na následujícím obrázku:



Obr. 5 : Popis funkce skriptů vypisujících projekty

Skripty pracující na bázi tohoto schématu jsou následující a liší se jen v maličkostech a hlavně v dotazech:

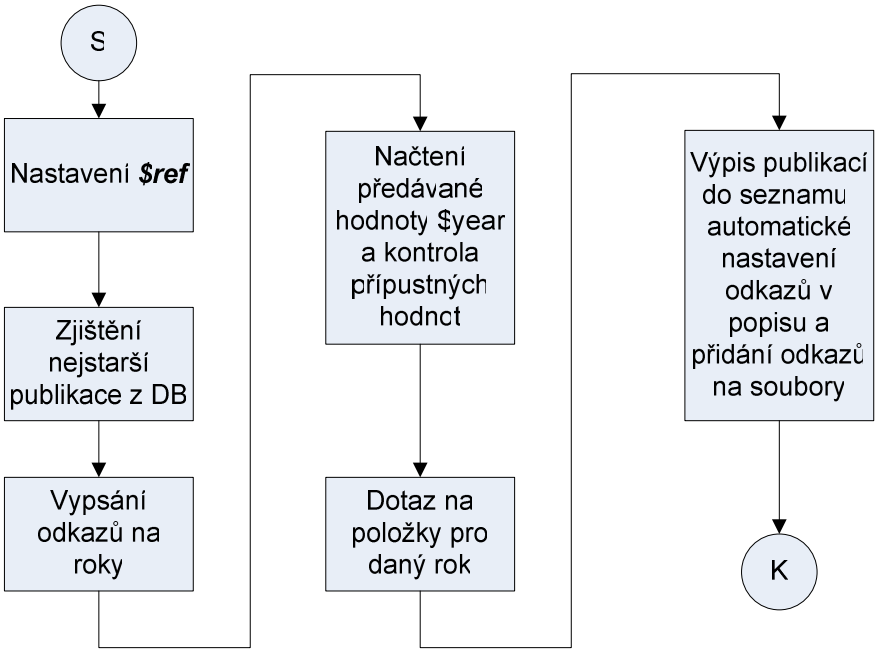
- **offered.php** – nabízené projekty
- **currproj.php** – běžící projekty
- **oldproj.php** – Tento skript se liší od předchozích dvou ještě tím, že neobsahuje odkazy na druhy projektu, protože u ukončených projektů jsme se rozhodli tuto informaci neudržovat.

Je zbytečné zde uvádět jejich kód. Můžete ho nalézt v uvedených souborech v kořenovém adresáři.

3.3.3. Publikace

Na této stránce najdete výpis publikací podle roku jejich vydání. Odkazy na roky vydání se automaticky aktualizují z databáze. Nejstarší rok je vždy ten, co je nejstarší záznam v databázi. Skript je koncipován tak, aby byl funkční i pro stovky let. V části popisu publikace se automaticky rozpoznávají odkazy pomocí regulárních výrazů začínajících na:

- http://
- www.



Obr. 6 : Popis funkce skriptu pro výpis publikací: pubs.php

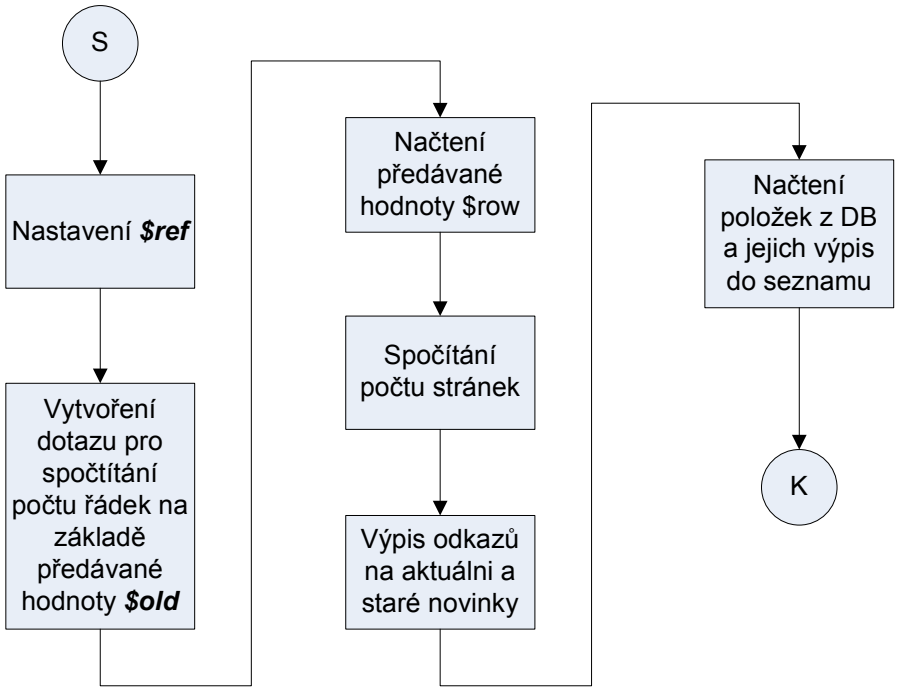
U každé publikace se můžou vyskytovat soubory ke stažení. Původní požadavek byl alespoň 2 soubory. Já jsem dal možnost více souborů, omezenou délkou součtu jejich jmen (ukládání do databáze). Upřesněno to je při popisu jejich nahrávání v příloze. Průběh skriptu je naznačen v blocích na obrázku č.6.

3.3.4. Novinky

Zde uživatel bude mít možnost nahlédnout do událostí, které skupinu potkaly nebo čekají. Stránka je rozdělená na dvě:

- **Aktuální novinky**
- **Staré novinky**

U každé novinky je uveden datum jejího zadání a datum, do kterého daná novinka platí. Po vypršení tohoto data se novinka začne zobrazovat jako stará. Dále je u každé novinky uveden její zadavatel. Novinka je rozdělena na dvě pomyslné části **Název** a **Popis**. V obou částech se mohou vyskytovat odkazy, které jsou automaticky rozpoznány. Průběh skriptu si můžete prohlédnout na následujícím obrázku:



Obr. 7 : Název Popis funkce skriptu pro výpis novinek: novinky.php

3.3.5. Administrační část

Tuto část najdete v adresáři **internal/** .Hlavním souborem je tu : **edit.php** . Je to rozcestník pro administrační část. Obsahuje tyto části:

- **Život projektu** – Zde lze vypisovat, zadávat a ukončovat projekty
- **Projekty k editaci** – Určeno pro editaci nebo mazání projektů. Pro přehlednost jsou rozděleny na nabízené, běžící a staré.
- **Novinky** – Možnost vypsání a editace novinek
- **Publikace** – Možnost vypsání a editace publikací.
- **Editace všech** – Tuto část vidí, jen ti členové VLSI skupiny, kteří jsou napsání v souboru **VLSIuser.php** v poli ***$admin***. Lze zde editovat všechny výše zmíněné položky, všech ostatních členů.

Dále jsou do této stránky přesměrovány všechny úspěšně či méně úspěšně proběhlé formuláře. Jsou zde vypisovány zprávy o úspěšnosti ukončeného formuláře. Skripty, v nichž nastane nějaká kritičtější chyba (nespojení s databází atd…), se ukončují.

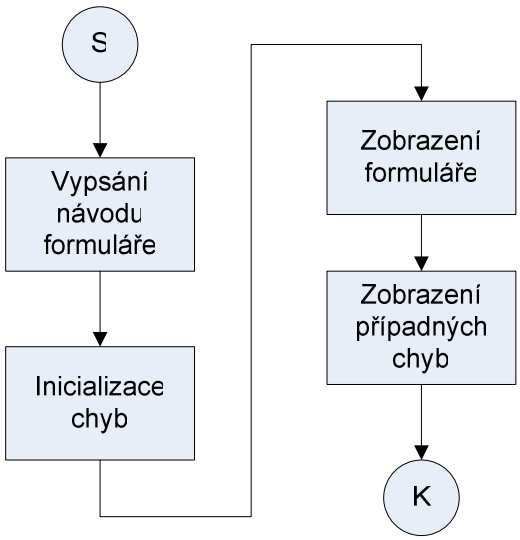
3.3.6. Formuláře

Formuláře jsem rozdělil do více kategorií a to podle:

- Zadávání nových dat
- Editace dat
- A dále jsou odlišeny formuláře pro **editaci všech**

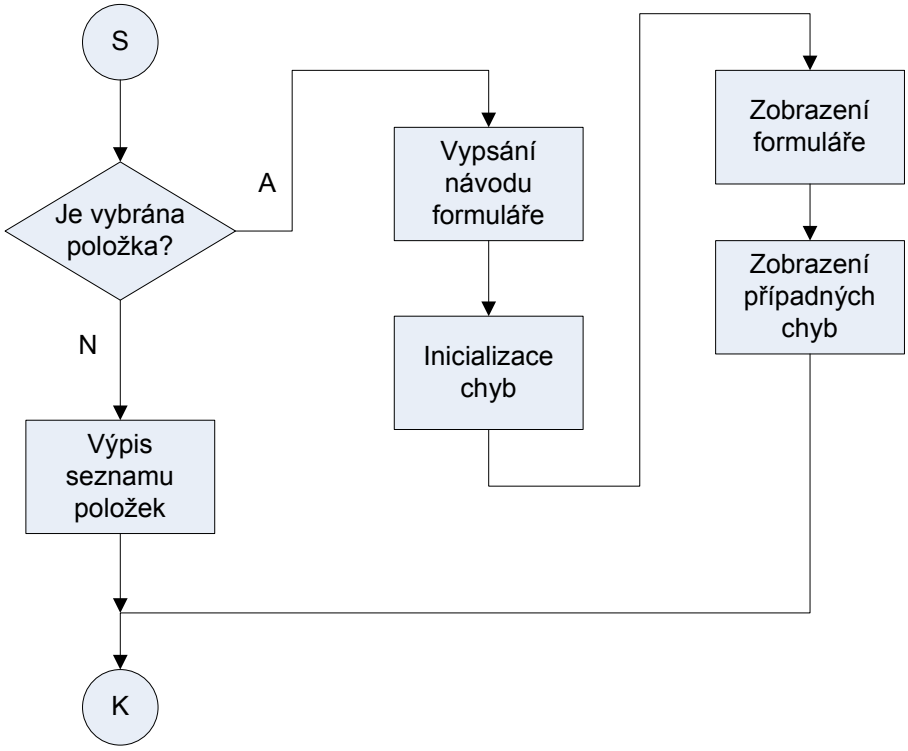
Každý formulář zde je tvořen dvěma soubory. Jedním, v němž je samotný formulář a druhým, který zpracovává data. Činnost skriptu lze odhadnout pomocí jména (popis v příloze).

Zadávací formuláře se chovají dle následujícího obrázku:



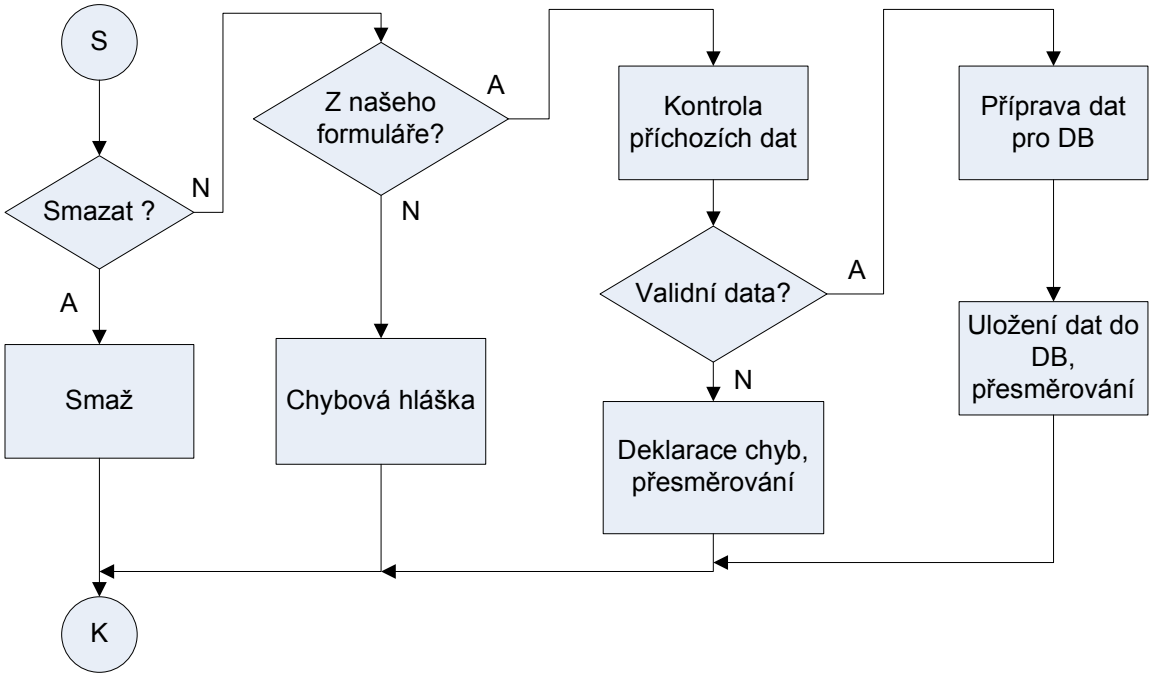
Obr. 8 : Činnost zadávacích formulářů

Editační formuláře se odlišují tím, že nejdříve si člověk musí vybrat položku (např. určitou publikaci, kterou zadal), kterou bude chtít editovat. Tudíž se při prvním zobrazení nezobrazí formulář, ale seznam položek možných k editaci.Činnost těchto skriptů je znázorněna na následujícím obrázku:



Obr. 9 : Činnost editačních formulářů

Skripty zpracovávající formuláře se chovají následovně (u skriptů zpracovávající zadávací formuláře je vynechána část smazat) :



Obr. 10 : Činnost skriptů zpracovávající formuláře

Předávání dat mezi skripty:

Předávání dat mezi skripty se může uskutečňovat pomocí různých metod. Pro nás přicházejí v úvahu metody **GET** a **POST.** Metoda GET předává hodnoty v odkazu a jsou tedy vidět v adrese v prohlížeči. Metoda POST předává hodnoty v těle požadavku. Z hlediska bezpečnosti se dá oběma metodám podstrčit jednoduše jiná data, takže se na straně přijímacího skriptu data vždy kontrolují na přípustné hodnoty. Pro všechny formuláře jsem použil metodu POST. Některé to přímo vyžadují (pro přenos souborů).

Formulář pro výpis nových projektů:

Tento formulář je ve skriptu s názvem **addproj.php** a zpracovávající skript má název **insertpr.php**. Ve formuláři je možnost zadat vše potřebné pro nabízený projekt. Typ projektu se vybírá jako roletové menu. V těchto skriptech není nic zvláštního a odpovídají výše popsaným obrázkům. Není teda potřeba je zde více rozebírat.

Formulář pro zadání projektů:

Naleznete ho pod jménem **reserve.php** a zpracovávající formulář jako **reserving.php**. Po zobrazení tohoto skriptu budete mít na výběr projekty, které jste vypsali. Po výběru projektu se vám zobrazí formulář, který umožňuje projekt přesunout do zadaných. Je tu možnost (CHECKBOX) nechat původní znění projektu jako nezadané. Na základě tohoto CHECKBOXU zpracovávající formulář rozhodne, zda se bude projekt jen aktualizovat nebo se vloží jako nový.

Formulář pro skončení projektů:

Skript má název **close.php** a jeho zpracovávající formulář **closing.php**. Dává na výběr z projektů, které jste zadali a po výběru projektu zobrazí formulář. Z návrhu tabulek v databázi vyplývá, že uzavření projektu má dvě části:

- Vložení projektu do tabulky skončených
- Smazání projektu z tabulky zadaných

Z toho vyplývá, že jsem musel použít dva dotazy na databázi. Kdyby ale nastal případ, že jeden dotaz se provede úspěšně a druhý ne, tak bychom nedosáhli toho, co chceme a v datech by byla chyba. Tudíž potřebujeme, aby se dotazy provedly oba, anebo ani jeden. Databáze Postgre umožňuje takovéto zacházení pomocí transakcí. Takže před prvním dotazem zahájíme transakci, provedeme oba dotazy a ukončíme transakci. V případě chyby jednoho z dotazů provádíme ROLLBACK- vrácení změn nazpět. Pro ostatní uživatele, při zahájení transakce, jsou data prezentována jako nezměněná, dokud není transakce ukončena.

Formuláře pro editaci projektů:

Jedná se o jeden formulář s názvem **editproj.php** a jeden zpracovávající skript **updproj.php**. Vedou na něj tři odkazy a v každém z nich je nastaveno, jaké projekty se mají editovat (nabízené, běžící, ukončené). Na základě tohoto se sestavují dotazy na příslušné tabulky. Jinak skript odpovídá obrázkům, které byly uvedeny na začátku této podkapitoly.

Formulář pro vkládání novinek:

Formulář má název **addnews.php** a zpracovávající skript **insnews.php**. V tomto formuláři se zadávají položky Název, popis a datum. Název a popis může obsahovat webové odkazy, které budou automaticky rozpoznány. Datum se zadává pomocí tří rolovacích menu den, měsíc a rok. Problém nastává při zadání špatného, či starého data. Skript si poradí se špatným formátem data, např. 30. února, a také rozpozná staré datum. Nejdou tedy vložit staré novinky. Zadavatel musí nastavit datum, alespoň na zítřek (výchozí hodnota). Datum se kontroluje pomocí následujícího kódu:

```
if ( Time() > MkTime (0,0,0, $to_month, $to_day, $to_year ) ) {
   $to_date = Date("Y-m-d", MkTime(0,0,0,date("m"),(date("d")+1),date("Y"))); }
```

Toto je ukázka kódu ze skriptu **addnews.php**. Tomuto kousku kódu předchází nastavení hodnot ***$to_month, $to_day, $to_year***, v závislosti na tom, zda byl formulář chybně vyplněn, nebo je otevřen poprvé. Funkce ***Time()*** vrací aktuální čas ve tvaru počtu sekund od začátku unixové éry (1.ledna 1970 00:00:00). Funkce ***MkTime()*** vrací čas ve stejném tvaru, ale počítá ho ze zadaných údajů. Tato funkce má tu výhodu, že si dokáže poradit i se špatným tvarem zadaného data. Funkce ***Date()*** umí zadaný čas funkcí ***MkTime*** naformátovat do požadovaného tvaru. Toho využíváme, abychom měli správný formát pro databázi. Do databáze ukládáme dvě data, kdy byla novinka vypsána a dokdy je platná.

Formulář pro editování novinek:

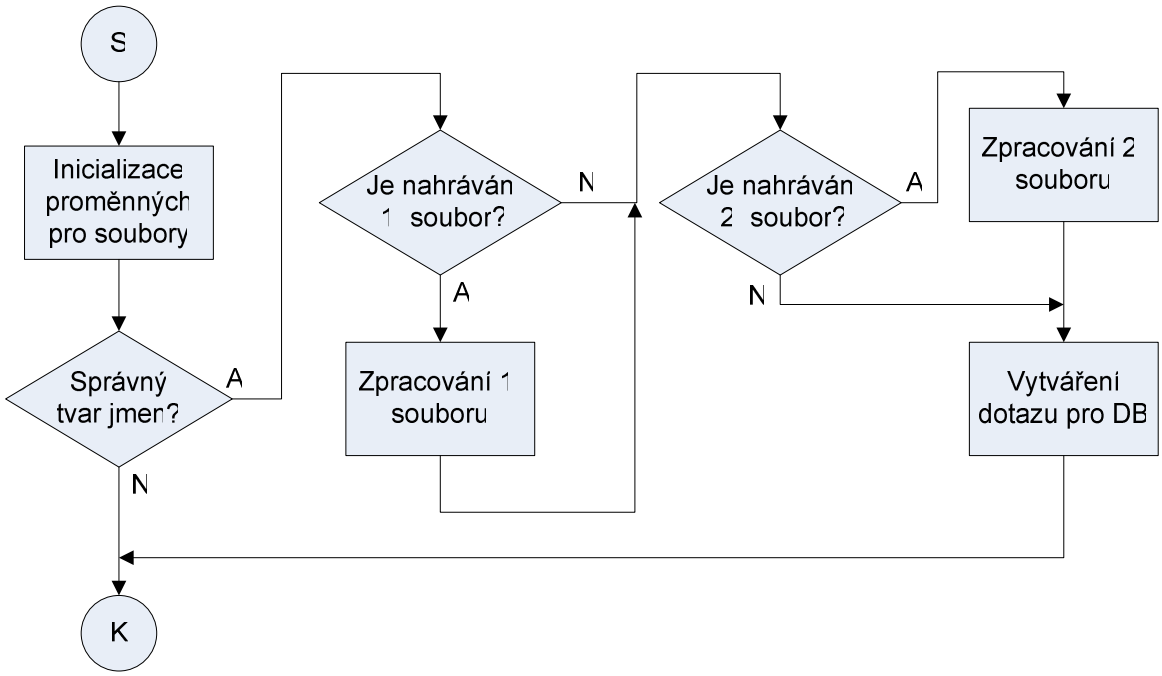
Formulář najdete ve skriptu **editnews.php** a zpracovávající formulář ve skriptu **updnews.php** . Skripty pracují na základě stejných funkcí a schémat (obr. 9 a 10) popsaných výše a proto není potřeba je zde rozebírat.

Formulář pro vkládání publikací:

Formulář naleznete v souboru **addpub.php** . U publikací lze zadávat položky Autoři, Popis, Rok a přiložit dva soubory. Aby šlo nahrávat soubory na server, musí být v tagu form uvedena položka ***enctype="multipart/form-data"***. Do položek Autoři a Popis lze zadávat odkazy, které budou automaticky rozpoznány. Ve formuláři jsou dvě pole pro možnost

přiložení souborů. Dále se zde zobrazuje informace o maximální velikosti přiložených souborů, které je možno na server nahrát metodou POST. Nastavení je získáno z php.ini pomocí funkce ***ini_get('upload_max_filesize')***.

Zpracovávající skript je **insertpb.php** . Funkce tohoto skriptu je stejná s ostatními až do zpracování souborů. To se děje na základě následujícího obrázku:



Obr. 11 :Zpracování souborů z formuláře

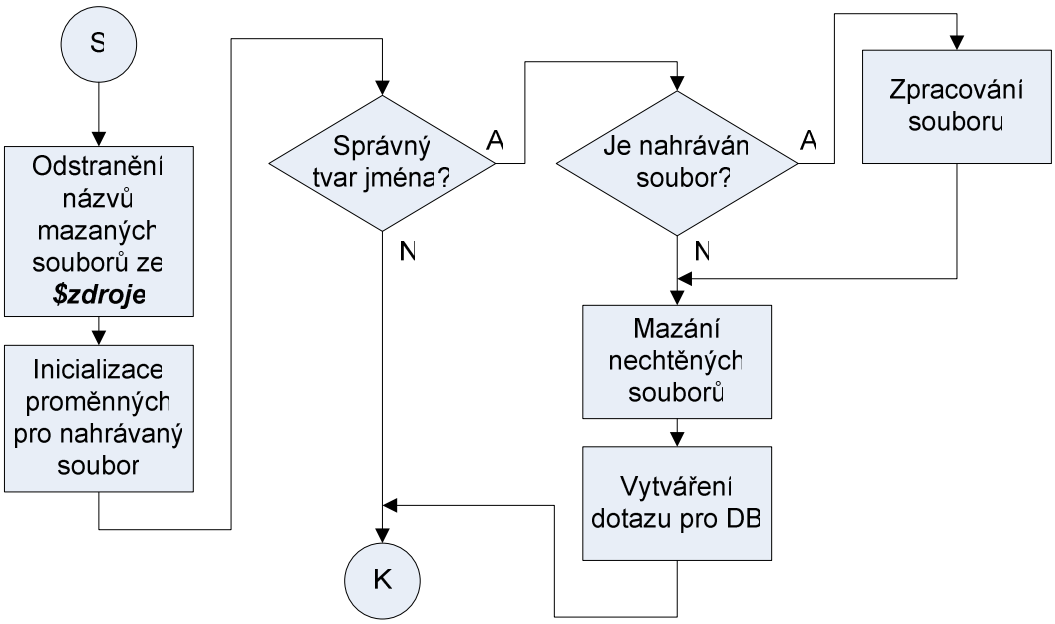
PHP nám poskytuje dvě funkce pro manipulaci s nahrávanými soubory metodou POST. Je nutné je požívat z bezpečnostních důvodů. Jsou to funkce ***is_uploaded_file( )*** a ***move_uploaded_file()***. Nejdříve zkontrolujeme, zda byl soubor nahrán metodou POST pomocí první funkce. Dále musíme zjistit, zda už v adresáři, kam budeme soubor ukládat, není jiný se stejným názvem. V takovém případě bychom ho museli přejmenovat.

Formulář pro editaci publikací:

Naleznete ho pod jménem ***editpubs.php***. V tomto formuláři je možnost editovat publikaci, smazat celou publikaci, či smazat nebo přidat soubory. Přidání souborů je provedeno stejně jako u předešlého skriptu pro zadávání publikace. Volba mazání souborů je ve formuláři znázorněna jako zaškrtávací políčko (checkbox) u odkazu na daný soubor. Jako jméno a hodnota checkboxu je jméno souboru a jeho koncovka bez tečky, kvůli spolehlivosti

odesílání hodnoty. Setkal jsem se s problémy u asociativních polí, když název obsahoval speciální znaky. Z tohoto důvodu nepřenášíme tečku v názvu checkboxu.

Zpracovávající formulář je **updpubs.php**. Oproti obrázku č. 10 je reskriptu navíc zpracování nahrávaného souboru a mazání nechtěných souborů. Ve skriptu je přidán kód chovající se dle následujícího obrázku:



Obr. 12 : Zpracování nahraného souboru a mazání nechtěných souborů

Odstranění názvů mazaných souborů ze ***$zdroje*** se provádí kontrolou, zda nám nepřišel z formuláře zaškrtnutý checkbox daného jména. Inicializace proměnných, kontrola správného jména a zpracování souboru je stejné jako u předcházejícího skriptu **insertpb.php** (popis naleznete v příloze) . Mazání souborů se provádí pomocí funkce ***unlink()***.

Formuláře pro editaci všech:

Tyto formuláře jsou stejné jako předešlé, až na to, že umožňují administrátorům uvedených v souboru **VLSIuser.php** v poli ***admin[]*** editovat projekty, novinky a publikace vypsané všemi členy VLSI skupiny. Tyto skripty tedy nemění položky zadavatel (příp.zadal). Na začátku těchto skriptů se nejdříve kontroluje, zda daný uživatel je uveden v poli ***admin[]***. Jak už bylo uvedeno na začátku této podkapitoly, názvy těchto skriptů začínají řetězcem „all“.

3.3.7. Bezpečnost a odolnost

Veřejná část:

Zde se naskýtá největší riziko možného útoku, protože k této části stránek bude mít přístup celý svět. Proto tu dodržuji určitá opatření:

- U všech předávaných proměnných metodou GET se kontroluje typ a hodnota, jestli je v rozmezí přípustných hodnot.
- Přihlašovací údaje do databáze jsou konstanty.
- Z této části se do databáze nic neukládá. Všechny dotazy jsou pevně dané až na položku OFFSET v SQL. Tato hodnota se samozřejmě kontroluje na typ a přípustné rozmezí.

Nejnebezpečnější na veřejné části je předávaná hodnota ***$page*** metodou GET. Z našeho pohledu to má být řetězec a používáme ho ke vkládání jiných souborů. To je nebezpečné z důvodu, že by daný útočník mohl do této proměnné podstrčit například URL se svým skriptem a ten by se pak vložil do toho našeho. Tomu jsem předešel tak, že proměnnou ***$page*** kontroluji regulárním výrazem, tak, aby se v ní nevyskytovaly žádné bílé, speciální a řídící znaky. V případě výskytu nějakého nechtěného znaku se načítá automaticky úvodní stránka. Dále kontroluji, zda soubor s daným jménem existuje. Tento kód si můžete prohlédnout v následující ukázce:

```
if(IsSet($_GET['page'])):{
    $kontrola='([[:space:]]|[[:punct:]]|[[:cntrl:]])+';
    if(ereg($kontrola,$_GET['page'])) {
        $url="main.php";}
    else {
        if(strcmp($_GET['page'],"Download")==0){
            $url="Download/index.php";}
        else{
            $url=$_GET['page'].".php";}}}
else:
    $url = "main.php";
endif;
if (!file_exists($url)){
    $url="main.php";}
require $url;
```

V této části kódu je ještě navíc podmínka, která kontroluje, zda uživatel nepožaduje sekci Download. Ta je v jiném adresáři a proto se v takovém případě přidává do proměnné ***$url*** cesta do tohoto adresáře.

Zabránění zobrazování částí našich stránek bez navigace:

Tohoto je docíleno jednoduchým kódem, kde ve stránce **index.php** nastavíme proměnnou ***$kod*** na nějakou hodnotu a tu pak kontrolujeme ve vkládaných souborech. Pokud by si uživatel chtěl zobrazit jen část stránek nebo si je vložit někam k sobě, tak se zobrazí jen bílá stránka, protože hodnota ***$kod*** nebude nastavena. Zobrazit si stránku může jen s navigací přes **index.php** .

Přístup k databázi:

Aby nedošlo ke změnám přihlašovacích hodnot k databázi, tak jsou uvedeny jako konstanty v souboru **dbsettings.php**. Dále v neveřejné části do databáze ukládáme různé položky zadané členy VLSI skupiny. Musel jsem ošetřit speciální znaky, které by způsobily jinou interpretaci dotazů, než jsem chtěl. Nejnebezpečnější je vkládání řetězců. U databáze Postgre se řetězce dávají do jednoduchých uvozovek. Nepředpokládá se ze strany VLSI členů žádný útok, ale pokud by se ve vkládaném textu objevila jednoduchá uvozovka, minimálně by to způsobilo chybu dotazu. A pokud by šlo opravdu o útočníka, tak by mohl za uvozovku dát středník, ukončit tím dotaz a napsat za něj jakýkoli svůj. Tomu zabráníme escapováním uvozovek. PHP je většinou nastaveno tak, že všechny proměnné předávané metodou GET, POST a COOKIE escapuje automaticky, nicméně není to pravidlem. Takže já pomocí funkce

$slashes=get_magic_quotes_gpc();

zjišťuji, zda je na serveru automatické escapování, pokud ne, tak ho před vkládáním dat do databáze provádím funkcí ***addslashes()***. Pokud si hodnoty předávají skripty vícekrát, například při chybně vyplněném formuláři, tak se naopak musí provádět funkce ***stripslashes()***, která je opakem funkce ***addslashes()***, aby nedocházelo k vícenásobnému escapování.

Pozn.: **Escapování** – Nahrazení znaků v řetězci, které jazyk SQL používá pro své potřeby a mohl by je interpretovat jinak než jsme zamýšleli, sekvencí znaků, které interpretuje správně. Například jednoduchá uvozovka se používá pro vymezení řetězce. Pokud by řetězec obsahoval jednoduchou uvozovku například ve jméně O´neil, tak by došlo ke špatné interpretaci řetězce. Proto se jednoduchá uvozovka nahrazuje sekvencí znaků: /´

Vícenásobné vložení jednoho formuláře do databáze:

Tato nechtěná situace může nastat ve více případech:

- Mačkání klávesy pro Reload stránky, čí pohybování se opakovaně v prohlížeči stránkami dopředu a dozadu (šipky u prohlížeče vlevo nahoře)

- Opakované mačkání tlačítka Odeslat ve formuláři při pomalé odezvě serveru.

Prvnímu případu se dá zabránit snadno a to tak, že se na zpracovávání dat z formuláře používáme jiný skript, než ve kterém je formulář a z něj se pak přesměrujeme jinam. Daný skript nemá tedy žádný textový výstup, a prohlížeč si ho nepamatuje, a proto nenastává vícenásobné vkládání hodnot do databáze.

Druhému případu se zabraňuje hůře. Je nutno podotknout, že nastane pouze při špatném připojení, při dlouhé odezvě. Zabraňuji tomu jednoduchým postupem, tak, že ve formuláři nastavím určitou proměnnou v SESSION a tu pak před vkládáním do databáze ruším. Druhý požadavek na vložení dojde do stejného místa ve skriptu a zjistí, že proměnná je již zrušená a skončí. Je to jednoduché opatření, ale má jednu nevýhodu, a to takovou, že jeden uživatel může naráz pracovat s jedním od každého formuláře. Existují i jiné postupy, například při otvírání formuláře se generuje unikátní proměnná, která se pak vkládá do databáze se všemi položkami. V databázi se pak kontroluje, zda hodnota už zde není. Tento způsob je však mnohem náročnější na databázi a jsou i problémy s efektivním generováním unikátních proměnných. Z těchto důvodů jsem zvolil jednodušší způsob se SESSION s uvedeným omezením.

Nahrávání souborů na server:

Při nahrávání souborů na server je důležité používat následující dvě funkce, které nám zajistí, že daný skript bude pracovat jen se soubory, které byly úspěšně nahrány metodou POST, aby se neukázněný uživatel nepokusil o manipulaci s jinými soubory. Jde o funkce ***is_uploaded_file()*** a ***move_uploaded_file()***.

3.4. JavaScript

Protože JavaScript nemusí běžet v každém prohlížeči, tak ho používám jen pro základní kontrolu formulářů na straně klienta, aby se ušetřila komunikace se serverem. Na straně serveru se formulář kontroluje vždy, takže i v případě nefunkčnosti JavaScriptu je zajištěna správná funkce webu. Zde je krátká ukázka:

```
var text_nazev = self.document.forms.addnews.nazev;
if (text_nazev.value.length==0) {
    text_nazev.focus();
    alert("Musíte vyplnit Název.");
    return false;}
```

Na prvním řádku do proměnné uložíme ukazatel na objekt ve struktuře stránky. V podmínce poté kontrolujeme jeho hodnotu na nenulovou délku. Pokud je délka nulová, tak

kurzor přesuneme na dané políčko a pomocí funkce ***alert()*** upozorníme na špatné vyplnění formuláře. Toto je jen ukázka kontroly jedné hodnoty. Povinné atributy se kontrolují na nenulovou délku a maximální délku. Nepovinné pouze na maximální délku.

4. Testování

4.1. Úvod

Testování aplikace jsem rozdělil na požadavky (requirementy) tak, aby pokryly pokud možno všechny body, které má aplikace splňovat. Výsledky testů jsem shrnul do tabulky. Prováděl jsem i „náhodné“ testy, kdy jsem se pokoušel napodobit různé cesty projektů, publikací a novinek od jejich zadání až po jejich smazání, či archivaci.

4.2. Požadavky

4.2.1. Veřejná část

- Req. 1 – Aplikace musí splňovat požadavek na jednoduchou a přehlednou grafickou úpravu
- Req. 2 – Ovládání (Navigace) je intuitivní a funkční
- Req. 3 – Aplikace je odolná vůči změnám hodnot předávaných metodou GET
- Req. 4 – Aplikace generuje validní HTML kód a správně se zobrazuje v převládajících prohlížečích na trhu
- Req. 5 – Aplikace generuje validní CSS kód a správně se zobrazuje v převládajících prohlížečích na trhu
- Req. 6 – Novinky jsou rozděleny graficky na aktuální a staré
- Req. 7 – Novinky se automaticky přesouvají do starých po vypršení jejich platnosti
- Req. 8 – U Novinek se automaticky rozpoznávají odkazy
- Req. 9 – Publikace jsou rozděleny podle roků vydání
- Req. 10 – Odkazy na rok vydání se generují automaticky
- Req. 11 – Automaticky se generují odkazy na soubory u publikací
- Req. 12 – Automaticky se rozpoznávají odkazy u publikací
- Req. 13 – Projekty jsou přehledně rozděleny na Nabízené, Běžící a Staré
- Req. 14 – U Projektů a Novinek funguje stránkování položek

4.2.2. Neveřejná část

- Req. 15 – Aplikace generuje validní HTML kód a správně se zobrazuje v převládajících prohlížečích na trhu
- Req. 16 – Aplikace generuje validní CSS kód a správně se zobrazuje v převládajících prohlížečích na trhu
- Req. 17 – Používání je dovoleno pouze členům VLSI skupiny
- Req. 18 – Aplikace jednoznačně rozlišuje členy VLSI skupiny

- Req. 19 – Členové mohou vypisovat projekty, publikace a novinky. Dále mohou zadat či ukončit běžící projekt
- Req. 20 – Editace projektů, publikací, novinek a zadání či skončení projektů je povoleno pouze tomu, kdo je zadal s výjimkou administrátorů
- Req. 21 – Aplikace umožňuje nastavit administrátory, kteří mohou editovat i cizí projekty, publikace, novinky a zadávat či končit projekty
- Req. 22 – U zadávání projektu je možné nechat původní znění projektu nezadané
- Req. 23 – U novinek se zadává datum její platnosti
- Req. 24 – Při vkládání publikace je možnost nahrát dva soubory
- Req. 25 – Při editaci publikace je možnost nahrát soubor
- Req. 26 – Při editaci je možnost mazat nahrané soubory
- Req. 27 – Do databáze se nesmí formulář vkládat opakovaně při pohybu historií vpřed a vzad v prohlížeči
- Req. 28 – Do databáze se nesmí formulář vkládat opakovaně při vícenásobném odeslání formuláře (vícenásobné zmáčknutí tlačítka odeslat)
- Req. 29 – Jsou ošetřeny speciální znaky (uvozovky) při ukládání dat do databáze.

4.3. Výsledky

Požadavky	Výsledky
Req. 1	v pořádku
Req. 2	v pořádku
Req. 3	v pořádku
Req. 4	v pořádku
Req. 5	v pořádku
Req. 6	v pořádku
Req. 7	v pořádku
Req. 8	v pořádku
Req. 9	v pořádku
Req. 10	v pořádku
Req. 11	v pořádku
Req. 12	v pořádku
Req. 13	v pořádku
Req. 14	v pořádku
Req. 15	v pořádku
Req. 16	v pořádku
Req. 17	v pořádku
Req. 18	v pořádku

Req. 19	v pořádku
Req. 20	v pořádku
Req. 21	v pořádku
Req. 22	v pořádku
Req. 23	v pořádku
Req. 24	v pořádku
Req. 25	v pořádku
Req. 26	v pořádku
Req. 27	v pořádku
Req. 28	v pořádku
Req. 29	v pořádku

Tab. 4 Výsledky testů

4.4. Závěr

Během vývoje této aplikace byly prováděny testy jednotlivých vyvíjených součástí a po jejich dokončení byly opět testovány jako celek. Jak je vidět z tabulky č.4, všechny provedené testy z požadavků proběhly v pořádku. Mými „náhodnými“ testy jsem se snažil pokrýt všechny možnosti, ale samozřejmě je nelze pokrýt všechny.

5. Uživatelská příručka

5.1. Uvedení do provozu

Zde je uveden postup, jakým by se mělo postupovat při uvádění aplikace do provozu:

- Založíme databázi a vytvoříme v ní tabulky pomocí SQL dotazů uvedených v souboru **tabulky.sql**
- V souboru **dbsettings.php** nastavíme přihlašovací údaje do databází
- V souboru **VLSIgroup.php** nastavíme konstantu ***pokolika*** – po kolika položkách se má zobrazovat stránkování
- Nahrajeme aplikaci do veřejné části webu
- U adresáře **Zdroj** nastavíme práva pro zápis
- Adresář **Internal** je určen pro běh v zabezpečené části serveru (minulá interní sekce webu) , a proto ho tam přesuneme
- V souboru **VLSIuser.php** nastavíme cesty u prvních 4 řádků kódu. (Místo **../** napíšeme cestu do veřejné části webu)
- V souboru **VLSIuser.php** nastavíme konstantu ***pokolika*** – po kolika položkách se má zobrazovat stránkování
- V souboru **VLSIuser.php** zrušíme komentářové značky u proměnné ***$ident***, tak aby kód vypadal následovně:

  //$ident="fiserp"; ---------- pro testování
  $ident=$REMOTE_USER;
- V souboru **VLSIuser.php** napíšeme do pole ***$admin*** loginy členů, kteří mají mít administrátorská práva (mohou editovat vypsané položky i ostatním členům) Například:

  $admin=array(1=>"fiserp");
- Aplikace je nyní plně funkční

5.2. Rady při potížích

5.2.1. Nejde čeština v databázi

Příznaky: Po zadání českých znaků do databáze se zobrazují jiné znaky.

Řešení: V souboru **dbconn.php** zrušíme komentářové značky /* a */, tím dosáhneme toho, že databáze bude komunikovat pomocí dané znakové sady.

5.2.2. Neukládají se nahrané soubory v publikacích

Příznaky: Po nahrání souborů na server se sice objeví na ně v publikacích odkazy, ale soubory v adresáři **Zdroj** nejsou.

Řešení: Nastavení práv pro zápis u adresáře **Zdroj**. Nastavení správné cesty v souboru **VLSIuser.php** .

5.2.3. Všichni členové VLSI jsou přihlášeni jako jeden uživatel

Příznaky: Všem členům po vstupu do neveřejné části se zobrazuje jméno **Ing. Petr Fišer**

Řešení: V souboru **VLSIuser.php** zrušíme komentářové značky u proměnné ***$ident***, tak, aby kód vypadal následovně:

//$ident="fiserp"; ---------- pro testování
$ident=$REMOTE_USER;

6. Zhodnocení

Cílem této práce bylo vytvořit bezpečnou aplikaci pro internetové stránky VLSI skupiny. Tohoto cíle bylo plně dosaženo pomocí PHP a databázového systému PostgreSQL. Aplikace umožňuje členům VLSI skupiny vypisovat, zadávat, ukončovat projekty a vypisovat novinky a publikace. K publikacím lze nahrát na server soubory nebo třeba v textu uvést odkazy, které jsou automaticky rozpoznány. Aplikace byla vytvořena tak, aby splňovala bezpečnostní požadavky, a to hlavně její veřejná část.

Možnost rozšíření této práce by mohla směřovat k přenositelnosti aplikace. Aplikace byla vytvořena přímo na míru a to zvláště databázová část, která vlastně neobsahuje údaje o členech VLSI skupiny. Databáze by tedy mohla být rozšířena o tabulku členů, případně i jiných položek v závislosti na potřebách dalších skupin. Během vývoje této aplikace vznikly na stránkách FEL (info336) oficiální stránky, které z části pokrývají tento projekt. Během vývoje jsme o tomto webu nevěděli a proto nám tu vznikla redundance. Další rozšíření by tedy mohlo být v propojení na databázi info336.

7. Závěr

Výsledkem této bakalářské práce jsou fungující webové stránky. Ty umožňují členům VLSI skupiny informovat veřejnost o jejich aktivitách, novinkách, projektech a podělit se v publikacích o výsledky jejich prací. Nabízí možnost vypisovat pro studenty projekty, dále je zadávat či ukončovat. Všechny tyto uvedené položky jsou udržovány v databázovém systému PostgreSQL. Dále v rámci této bakalářské práce byla navržena jednotná grafická úprava prezentace s ohledem na přehlednost a jednoduchost použití. Kód stránek byl napsán dostatečně jednoduše a přehledně, aby byla usnadněna jeho případná aktualizace. Cílem této práce nebylo vytvoření obsahu stránek. Ten byl převzat ze staré verze.

8. Literatura a použité vývojové programy

[1] Petr Staníček, Kompletní průvodce CSS, Computer Press Brno 2003

[2] Jiří Bráza, PHP 5 začínáme programovat, Grada Praha 2005

[3] Hugh E. Williams & David Lane, PHP a MySQL, Computer Press Praha 2002

[4] Bruce Momjian, PostgreSQL Praktický průvodce, Computer Press Brno 2003

[5] Online manuál PHP, http://cz.php.net

[6] Online validátor HTML , http://validator.w3.org/

[7] Online validátor CSS, http://jigsaw.w3.org/css-validator/

[8] Wamp5 (instalace Apache + PHP 5 a MySQL), http://www.wampserver.com/

[9] PostgreSQL 8.1, http://www.postgresql.org/

[10] PHP Designer 2005, http://www.mpsoftware.dk/

9. Seznam obrázků

10. Seznam Tabulek

11. Přílohy

11.1. Rozdělení skriptů

Skripty jsou rozděleny dle jejich umístění a funkčnosti na veřejnou a neveřejnou část.Veřejné skripty najdete přímo v kořeni adresáře a skripty pro neveřejnou část najdete ve složce ***internal***.

11.2. Práce s databází

Základní soubory pro práci s databází mi byli dány k dispozici. Jednalo se o několik souborů, které zajišťují autentizaci uživatele na serveru service. Využil jsem z nich několik souborů a najdete je ve veřejné části ve složce SERVICE_PHP:

- **err.php**
- **pg.php**

V souboru **err.php** je definice objektu, který se stará o zpracování chyb nastalých při komunikaci s databází. Tento soubor zůstal nezměněn až na jednu výjimku. Jako komentář jsem označil část skriptu, který zapisuje nastalé chyby do souboru. V případě potřeby případného doladění, se může tato změna vrátit zpět.

V souboru **pg.php** je definice objektu pro práci s databází na serveru service. Funkčnost skriptu nebylo zapotřebí nijak výrazně měnit, protože na serveru service úspěšně běží. Změnil jsem tam několik drobností pro vlastní potřebu, a to následující věci:

- Zrušil jsem vytváření objektu chyby při chybné komunikaci s databází. Je to proto, že připojovat k databázi se bude i ve veřejné části a tímto zajistíme, že se uživatel při případné nefunkčnosti databáze či jiného problému nedozví nic o struktuře naší databáze z chybových hlášek. Vytváření objektu chyby je označeno jako komentář, takže se dá pro případné ladění snadno použít.
- Zrušil jsem výpis dotazu při jeho špatné formulaci či chybě DB, a to ze stejných důvodů jako výše, aby se uživatel nedozvěděl nic o struktuře DB.

Soubor s nastavením databáze je ve veřejné části a jmenuje se **dbsettings.php.** Obsahuje konstanty pro připojení k databázím jako jsou hesla, loginy a jména databází. Tyto položky jsou zde jako konstanty proto, aby nemohlo dojít k jejich změně, při nesprávném použití či útoku.

Dále jsou zde důležité následující dva soubory a to **VLSIgroup.php** ve veřejné části a **VLSIuser.php** v neveřejné části. Oba soubory zajišťují získání základních informací

z databáze na service o členech VLSI skupiny. Najdete v nich i funkce a konstanty, které budou popsány později.

Soubor **VLSIgroup.php** zajišťuje získání jmen VLSI členů. Popis struktury tabulek v DB na service najdete v souboru **SERVICE_PHP/auth.txt**. Dotaz na DB je trochu delší a komplikovanější a proto ho zde uvádím:

```
SELECT u1.username,u1.fullname
FROM usernames u1
 WHERE u1.uid IN(        SELECT g1.uid
                         FROM groups g1
                         WHERE g1.gid=(          SELECT g2.gid
                                                 FROM groupnames g2
                                                 WHERE g2.groupname='VLSI'));
```

Dotaz je tvořen poddotazy, protože ve většině případů jsou rychlejší než náročné spojování tabulek s mnoha řádky. V nejnižším dotazu v této struktuře zjišťujeme primární klíč jména skupiny. V druhém dotazu vybereme klíče uživatelů, kteří do dané skupiny patří. A ve třetím nejvrchnějším dotazu se ptáme na jména těch, kterým dané klíče patří. Výsledek dotazu je uložen do asociativního pole ***$jmena[]*** pro pozdější použití ve skriptech.

Soubor VLSIuser.php obsahuje výňatek kódu z dodaných skriptů ze souboru **SERVICE_PHP/user.php** . Jedná se o získání jména z databáze na service po přihlášení do interní sekce. Jméno je uloženo do proměnné ***$user*** pro pozdější použití.

Připojování k naší databázi ve skriptech:

Ve skriptech se objevuje následující část kódu, která vkládá do skriptu soubor s připojením k naší databázi:

```
if(!file_exists("dbconn.php")){die("Nelze se pripojit k db.");} require "dbconn.php";
```

Kontroluje se, zda daný soubor existuje a poté se vkládá do kódu obsah tohoto souboru. V souboru je kód pro připojení k naší databázi. Za ním je část kódu, který zajišťuje, aby databáze s námi komunikovala v naší znakové sadě. Tato část kódu je jako komentář, protože, aby Postgres mohl převádět znakovou sadu, musí mít danou databázi vytvořenou s rozšířenou znakovou sadou, např. UTF-8. V případě, že dotaz nefunguje, tak má Postgres databázi vytvořenou v ASCI podle nastavení daného počítače, což nám nevadí a převod nepotřebujeme, protože na daném počítači je čeština. Při uvádění projektu do provozu zjistíme či vyzkoušíme nastavení serveru a případně zrušíme komentářové značky u daného kódu.

11.3. Části kódu použité ve více skriptech a jejich popis

Vytváření odkazů – nastavení proměnné *$ref*:

Pro správnou funkčnost webu jsem musel zajistit, aby zůstával zachován jazyk, který si uživatel zvolil. To zajišťuje následující ukázka kódu:

```
$ref=$_SERVER['HTTP_REFERER'];
if(false===strpos($ref,"indexen")){$ref="index";}else{$ref="indexen";}
```

Na prvním řádku ukládáme do proměnné ***$ref*** adresu stránky, ze které uživatel přišel. Následně ji porovnáváme, zda obsahuje řetězcovou hodnotu ***indexen*** , což by značilo, že uživatel přišel z anglické části. Dle uvedeného zjištění ukládáme do ***$ref*** danou hodnotu použitelnou pro vytváření odkazů.

Tento princip nemusí být úplně vždy funkční. Uživatel si může na prohlížeči třeba zakázat posílání informace v proměnné ***HTTP_REFERER***. Dnes používané prohlížeče však tuto proměnnou odesílají. Tuto část skriptu používám jenom při zobrazování stránek, které jsou v databázi a jsou v češtině (projekty, publikace a novinky), takže informace, které uživatel požaduje, budou stejně v českém jazyce, anglická zůstane pouze navigace.

Funkce pro stránkování:

Pro lepší přehlednost stránek jsem vytvořil funkci, která vypisuje stránkovací odkazy. Ve skriptech můžete narazit na tuto funkci s více parametry, ale jako základní potřebuje hodnoty:

- **aktuální stránka**
- **celkový počet stránek**
- **stránka na kterou se odkazuje**

Základní verzi si můžete prohlídnout zde:

```
function rolovani($row,$pocetlistu,$ref){
if ($pocetlistu>1){
  if ($row==2) {echo(" <a href=\"$ref.php?row=1\"> &lt;&lt; </a> ");}
  if ($row>2) {echo(" <a href=\"$ref.php?row=1\"> &lt;&lt; </a>");$pom=$row-1;
              echo(" <a href=\"$ref.php?row=$pom\"> &lt; </a> ");}
  echo(" <span style=\"color:red;\">$row/$pocetlistu </span>");
  if ($row+1==$pocetlistu){echo("<a href=\"$ref.php?row=$pocetlistu\"> &gt;&gt; </a> ");}
  if ($row+1<$pocetlistu) {$pom=$row+1;echo(" <a href=\"$ref.php?row=$pom\"> &gt;</a> ");
                           echo(" <a href=\"$ref.php?row=$pocetlistu\"> &gt;&gt;          </a>"); }
              }                    }
```

První podmínka řeší případ, že je počet listů menší jak 2. V takovémto případě se rolovací odkazy nezobrazují. Dále jsou dvě podmínky pro listování dopředu, pak následuje výpis aktuální stránky / celkový počet stránek a dvě podmínky pro listování dozadu. Myšlenka této funkce je dle mého názoru jasná a není potřeba ji dále rozebírat.

Kód pro spočítání počtu stránek a vytvoření dotazu:

Tento kód je delší a proto ho zde nebudu uvádět. Můžete si ho prohlídnout například ve skriptu **offred.php** ve veřejné části. Uvedu pouze postup v bodech, který používám:

- připojení k DB a provedení dotazu na spočtení počtu položek pro daný výpis
- načtení a kontrola čísla zobrazované stránky, zda není mimo rozsah přípustných hodnot
- spočtení celkového počtu listů
- Vytvoření nebo upřesnění dotazu (přidání klauzule LIMIT a OFFSET)
- Výpis rolování

Kód pro automatickou detekci odkazů:

Jedná se o regulární výraz, pomocí kterého ve vybraných textech hledáme odkazy a dáváme je do tagů. Nejdříve si ukážeme kód, který na to používám:

```
$mu=htmlspecialchars($data["nazev"]);
$mu = EregI_Replace("(http://[^]+\.[^]+)", " <a target=\"_blank\" href=\\1>\\1</a>", $mu);
$mu = EregI_Replace("[^/](www\.[^]+\.[^]+)", " <a target=\"_blank\" href=http://\\1>\\1</a>", $mu);
```

V proměnné ***$data["nazev"]*** máme data načtená z databáze. Pomocí funkce ***htmlspecialchars( )*** escapujeme případné nežádoucí znaky a ukládáme do proměnné ***$mu*** . Funkce ***EregI_Replace*** nahradí část řetězce vyhovující regulárnímu výrazu řetězcem, v němž jsou tagy pro odkaz a původní část řetězce představovaná metaznakem ***//1*** . Tuto funkci použijeme dvakrát, jednou pro vyhledání řetězce začínající na **http://** a podruhé na začínající na **www.** Odkazy jsou nastavené tak, aby se otvíraly v novém okně. Tímto je daná položka připravena pro výpis do html stránky.

11.4. Publikace

Skript si můžete prohlédnout v kořenovém adresáři a má název **pubs.php**. Z obrázku číslo 6 je jeho funkce a průběh zřejmý. Za zmínku může stát pouze část kódu pro kontrolu příchozích hodnot ***$year*** :

```
if (!isset($_GET['year']))
    {$year=((int)date("Y"));}
else
    {$year=$_GET['year'];}
$year=(int)$year;
if ( ($year>date("Y")) || ($year<$minrok) )
    {$year=((int)date("Y"))-1;}
```

V první podmínce nejdříve zjišťujeme, zda vůbec byla nějaká hodnota předána. V případě, že ne, tak to znamená, že skript byl načten poprvé a ***$year*** nastavujeme na aktuální rok. V případě, že nám nějaká hodnota přišla, tak si ji ukládáme a přetypujeme na *int.* Dále kontrolujeme, zda je hodnota v rozmezí nejstaršího roku, zjištěno z DB a aktuálního roku. Pokud není, tak ***$year*** nastavujeme na aktuální rok.

11.5. Novinky

Skript si můžete prohlédnout v kořenovém adresáři a má název: **novinky.php** . Za zmínku stojí pouze část pro vytváření dotazu pro spočítání počtu řádek na základě předávané hodnoty ***$old*** :

```
if ( isset ($_GET['old'])  &&  $_GET['old']==1 )
   { $where=" WHERE to_date < NOW()";$stare="stare";$old=1; }
else
   { $where=" WHERE to_date >= NOW()";$stare="";$old=0; }
$dotaz="SELECT COUNT(*) FROM board".$where;
```

Nejdříve kontrolujeme, zda nám přišla nějaká hodnota a zda má definovanou hodnotu. Podle toho sestavíme klauzuli **where**, kde používáme funkci **NOW()**, která v databázi vrací datum v námi používaném formátu. Dále ještě nastavíme hodnotu ***$old*** pro další činnost skriptu a sestavíme dotaz pro databázi.

11.6. Popis názvů formulářů

Skripty začínající na:

- Add – zadávací formulář
- Ins – zpracování zadávacího formuláře
- Edit – editační formulář
- Upd – zpracování editačního formuláře
- All – formuláře a zpracování pro editaci všech

Zbytek jména obsahuje zkratku podle činnost např.: pr –projekty, pb- publikace atd… Výjimku tvoří formuláře pro zadání a ukončení projektu. Ty se jmenují reserve a close a zpracování těchto formulářů dělají stejnojmenné soubory s příponou –ing.

11.7. Zobrazení chyb ve špatně vyplněném formuláři:

Skript zpracovávající formulář, v případě, že zjistí při kontrole, že byl chybně vyplněn, tak data vrátí formuláři. Před formulářem se pak objeví proč je formulář špatně vyplněn. Ve formulářovém skriptu je vypisování chyby zajištěno například tímto kódem:

```
if ( isset($_GET['chybinazev']) && $_GET['chybinazev']==true )
   {echo "<br /><b><font color=\"red\">Musíte vyplnít políčko název</font></b>";}
```

Pokud formuláři byly vráceny hodnoty, a je nastavena daná proměnná, tak se vypíše příslušná chyba. Každá chyba má tedy svou proměnnou. Na straně zpracovávacího skriptu se kontrola provádí následujícím kódem:

```
$is_chyba=false;
if ( isset($_POST['nazev']))
   { $nazev=trim($_POST['nazev']); }
else
   {$nazev="";}
if ( strlen($nazev)==0 )
  { $is_chyba=true;$chyba.="chybinazev=true";}
if( $slashes==1 )
   {$chyba.="&nazev=".stripslashes($nazev).";}
else
   {$chyba.="&nazev=$nazev}
if ($is_chyba) { header ("Location: xxx.php?".SID."&$chyba");exit; }
```

Toto je příklad, kdy se kontroluje jedna věc a to název na nenulovou délku. Samozřejmě v našich formulářích se kontrolují všechny položky na nulovost, max.délku, aj. Na prvním řádku kódu jsme nastavili, že chyba zatím není. Dále načteme do proměnné $nazev předávanou hodnotu a pokud žádná nepřišla, tak ji nastavujeme na prázdný řetězec. Následuje samotná kontrola nenulovosti, při její nesplnění se nastaví proměnné, že nastala chyba a jaká. V podmínce $slashes==1 se do proměnné $chyba ukládají názvy a původní hodnoty a pokud nastala nějaká chyba, tak se odesílá nazpátek formuláři. Co znamená podmínka $slashes==1, bylo vysvětleno v části bezpečnost a odolnost. Před vypsáním formuláře při inicializaci probíhá ještě následující kód:

```
if ( isset($_GET['nazev']))
   {  if ($slashes==1)
       {$nazev=htmlspecialchars(stripslashes($_GET['nazev']));
     }else
       {$nazev=htmlspecialchars($_GET['nazev']);}}
 else
   { $nazev=""; }
```

V tomto kódu se rozhoduje, co se zobrazí ve formuláři. První podmínka rozlišuje, zda přišla vrácená data. U editačních skriptů se v části else nedává prázdný řetězec, ale hodnota z databáze. V případě, že je chyba a data byla vrácena, tak se upravují, aby se dala zobrazit.

11.8. Publikace - ošetření souborů

Inicializace proměnných pro soubory (obrázek č.11) je pouze získání informací o souboru (jméno, jméno na serveru, velikost), pokud je nějaký nahráván. Představuje ho následující kód pro každý soubor:

```
if ( isset( $_FILES["soubor1"]["name"]) ){
   $soubor_name1=$_FILES["soubor1"]["name"];}
if ( isset( $_FILES["soubor1"]["tmp_name"])) {
   $soubor1=$_FILES["soubor1"]["tmp_name"];}
if(isset($_FILES["soubor1"]["size"])){
   $soubor_size1=$_FILES["soubor1"]["size"];}
```

Správný tvar jmen se kontroluje pomocí regulárního výrazu, ve kterém sledujeme bílé, speciální a řídící znaky. Kontrolovat ovšem musíme pouze jméno souboru bez přípony, takže nejdříve oddělíme název, viz následující kód:

```
$kontrola='([[:space:]]|[[:punct:]]|[[:cntrl:]])+';
$lastind1=strrpos($soubor_name1,".");
$name1=substr($soubor_name1,0,$lastind1);
if ( ereg($kontrola,$name1)) {
   header ("Location: addpub.php?stav=6&".SID."&$chyba");exit;}
```

Na prvním řádku definujeme regulární výraz. Na druhém zjišťujeme pozici posledního výskytu tečky a na třetím oddělujeme název od přípony. V podmínce pak kontrolujeme název pomocí regulárního výrazu. V případě špatného názvu přesměrujeme na formulář.

Ukázka práce se soubory:

```
if (  file_exists(soubory_pubs.$soubor_name1))  {
    $hledej=1;
```

```
$konc=strrchr($soubor_name1,".");
$lastind=strrpos($soubor_name1,".");
$name=substr($soubor_name1,0,$lastind);
$jm=$name.$hledej;
while ( file_exists(soubory_pubs.$jm.$konc)){
    $hledej++;
    $jm=$name.$hledej; }
$soubor_name1=$jm.$konc; }
```

Na prvním řádku kontrolujeme, zda soubor se stejným jménem již neexistuje. V případě že ano, tak do ***$hledej*** ukládáme hodnotu, která se bude přidávat na konec jména souboru. Dále oddělíme koncovku a jméno souboru a v proměnné ***$jm*** vytvoříme nové jméno souboru. Ve smyčce pak měníme přidávanou část do jména a kontrolujeme, zda soubor neexistuje. Při nalezení neexistujícího jména se smyčka ukončí a celý název souboru se uloží do proměnné ***$soubor_name1***. Pro přesunutí souboru do našeho adresáře použijeme výše zmíněnou funkci ***move_uploaded_file()***.

Po nahrání souborů na server vytváříme dotaz a do proměnné ***$zdroj*** ukládáme názvy souborů oddělené **#**. Před vložením do databáze zkontrolujeme, zda řetězec v ***$zdroj*** není moc dlouhý a případně mažeme nahrané soubory a přesměrujeme na formulář.

11.9. Popis obsahu CD

- Adresář **Pages** – vytvořená aplikace v této bakalářské práci. Při jejím uvádění do provozu postupujte dle návodu v Uživatelské příručce. Běžící aplikaci můžete otestovat na internetu. Odkazy jsou uvedeny v kapitole Testování
- Adresář **Postgre** – instalace databázové aplikace PostgreSQL 8
- Adresář **Wamp** – instalace Apache serveru s PHP pr Windows
- Adresář **PHP Designer** – instalace mnou používaného vývojového prostředí pro PHP