VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

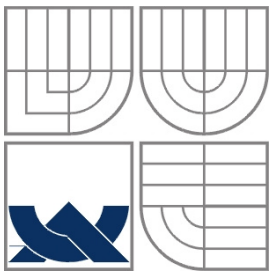
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

REDAKČNÍ SYSTÉM WEBARCHITEKT

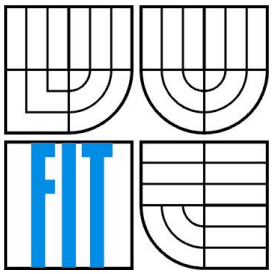
BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE LADISLAV SMOLA
AUTHOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

REDAKČNÍ SYSTÉM WEBARCHITEKT

CONTENT MANAGEMENT SYSTEM WEBARCHITEKT

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

Ladislav Smola

VEDOUCÍ PRÁCE
SUPERVISOR

Prof. Ing. Tomáš Hruška, CSc.

BRNO 2009

Abstrakt

Tato práce se zabývá tvorbou obecného redakčního systému, který funguje na bázi AJAX. Systém je modulární, ale zároveň založený na pluginech, které tvoří jeho jádro.
V první části se zabývám redakčními systémy a jazyky, které se nejčastěji používají pro tvorbu. Jsou zde uvedeny také knihovny a doplňky, které usnadňují programování.
Řešení problému je popsáno z části teoreticky, ale z větší části přímo pomocí objektového modelu, který jsem použil. Jsou popsány jednotlivé metody pro vyjádření funkčnosti jádra systému a dále pluginy, které půjdou k tomuto systému připojit.
Další vývoj je rozebrán na konci a jak je uvedeno, bude se řídit hlavně projekty na které bude tento systém dále použit.

Klíčová slova

CMS, redakční systém, šablony, šablonový systém, Smarty, PHP, JScript, AJAX, WYSIWYG

Abstract

This thesis concerns universal content management system, which works on AJAX basis. System is modular and based on plugins, which forms its core.
In first part, I am concern with CMS and programming languages, which are used for programming in most cases. There are also mentioned libraries and add-ons, which simplify programming itself.
Problem solution is described theoretically at the beginning, then I describe the object model itself. The methods of certain classes are described to express the functionality of system core and then a describe plugins which can be add to core.
Future development is mentioned at the end. It will be controlled mostly by the projects, on which will be this system apply next.

Keywords

CMS, content management system, templates, template engine, Smarty, PHP, JScript, AJAX, WYSIWYG

Citace

Smola Ladislav: Redakční systém WebArchitekt. Brno, 2009, bakalářská práce, FIT VUT v Brně.

Redakční systém WebArchitekt

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Prof. Ing. Tomáše Hrušky, CSc.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jméno Příjmení
Datum

Poděkování

Rád bych vyjádřil poděkování Prof. Ing. Tomáši Hruškovi, CSc za odborné vedení a rady, kterými mě podporoval a také za trpělivost.

Obsah

Úvod

V dnešní době je na trhu velké množství jak komerčních tak freeware CMS (Content management system). Důvodů proč tvořit vlastní je mnoho. Například pro menší firmu je levnější vytvořit vlastní systém, který lze přizpůsobovat podle zakázek. Náklady na koupi systému a školení programátorů mohou přesáhnout náklady na vývoj CMS vlastního. Použití freeware je v tomto případě lepším řešením, za předpokladu že systém je odladěný, zdokumentovaný a rozšiřitelný.

Pro tvorbu vlastního CMS jsem se rozhodl proto, že není zatím žádné freeware řešení přes AJAX. Jelikož se CMS začínají využívat ke každodenní práci ve firmách, bylo mým záměrem se pokusit zrychlit práci v systému a udělat ji více intuitivní. Tedy celý systém udělat AJAX technologií, což se v podstatě začíná blížit desktopové aplikaci. A pomocí drag & drop objektů práci zjednodušit. Načítat znovu celou stránku je poměrně zdlouhavé, navíc při použití frameworků, pak inicializace probíhá příliš dlouho, je tedy důležité vše udělat jen při zpuštění aplikace a dále pracovat jen s menšími částmi systému. Načítání takovýchto částí pak probíhá asynchronně.

Dalším důvodem je také zjednodušit práci programátorům. Tedy snaha o prostředí kde jdou vytvořit obecné položky, závislé jen pluginech, které jsou naprogramované(obrázkové galerie, diskusní forum, obecné formulářové prvky), tak aby šlo takto vytvořenou položku umístit na libovolné místo do systému. Struktura je pak je pak analogií klasického adresář, soubor.

V tomto dokumentu budu rozebírat dostupné technologie, s důrazem na co největší využití freeware doplňků, což výrazně snižuje náklady na celkový systém a zajišťuje pravidelné aktualizace.

Poté se budu snažit nastínit cíl mé práce stručným popisem požadované funkčnosti CMS.

Nakonec uvedu podrobné řešení tohoto problému, s ohledem na další vývoj mého CMS.

1 Stav problematiky

Redakční systémy se rychle vyvíjí, reagují na nové technologie. Na trhu je různých systémů velice mnoho. Já jsem se zaměřil na tvorbu systému přes AJAX, s tím že používám nejběžnější technologie, které jsou zároveň na serverech nejvíce podporované a takovýto hosting stojí nejméně. Jmenovitě jsou to kombinace PHP+MySQL, šablonový systém Smarty a mnoho dalších knihoven a doplňků, které jsou zdarma.

1.1 Redakční systémy

CMS existuje na webu celá řada. Většinou se jedná o systémy PHP + MYSQL. Ovšem freeware systém postavený celý na AJAX se mi najít nepodařilo.Provedu tedy porovnání s jedním komerčním, který mě zaujal. Při tvorbě svého systému jsem vycházel spíše z vlastních zkušeností, snažil jsem se zautomatizovat některé monotónní práce jako práce s položkami(stálé stránky, produkty), propojování pluginů se stránkami atd..

Pro srovnávání ještě uvedu co patří mezi klasické funkce CMS. Je to většinou:

- Tvorba a správa dokumentů. Měla by být podpora WYSIWYG editoru.
- Nastavení přístupových práv. Identifikace uživatelů. Popřípadě i funkce workflow atd.
- Správa galerií, obrázků a jiných souborů.
- Podpora jazykových verzí dokumentů.

Jeden ze známých CMS je například **WebNode**. Obsahuje úpravu stránek přes AJAX přímo ve frontendu, vše je drag & drop. Toto je ovšem vhodné pouze pro prefabrikované řešení. Tedy k dispozici jsou sady CSS, které se obměnují, případně vytváří pro nové klienty. Pokud chce klient ovšem úplně novou originální grafiku, bylo by velice zdlouhavé propojit nové šablony s tímto systémem. Proto jsem se rozhodl pro úpravu stránek v administraci. Při tomto řešení lze taktéž použít už připravený frontend, potom je důležité zachovat stejnou strukturu administrace a frontendu tak aby byla práce přehledná.

1.2 Technologie

V této kapitole rozeberu všechny použité programovací jazyky, freeware knihovny a ostatní prvky co jsou použity v mém CMS.

1.2.1 XHTML + CSS

Krátký popis XHTML a CSS, výhod používání standardů a krátké pojednání o optimalizaci pro vyhledávače.

Deset největších výhod tvorby kódu podle standardů.

1. ***Zobrazení v různých klientských aplikacích.***
   Stejná část takto naformátovaného obsahu může být lehce zobrazena v široké škále prohlížečů a zařízení (v klientských aplikacích), jako jsou webové prohlížeče, osobní asistenti – PDA, mobilní telefony a čtecí zařízení určená nevidomý. Pro každý prohlížeč či zařízení stačí pouze vytvořit jinou šablonu stylu.
2. ***Zvýšení výkonu.***
   Takto vytvořené stránky jsou jednodušší (velikost výsledného souboru je menší), a proto se také rychleji načtou. Důvodem je, že váš obsah potřebuje pouze minimální úpravu struktury. Pokud budete chtít dodržet standardy, odstraníte všechny značky, které tvořily formu prezentace na všech vašich stránkách, a nahradíte je jedinou šablonou kaskádových stylů. Jak se zanedlouho přesvědčíte, může jedna šablona stylů vytvořit formu na každé stránce vaší webové prezentace, soubor s kaskádovými styly se však načte pouze jednou.
3. ***Správné zobrazení ve všech prohlížečích.***
   S malým úsilím můžete dosáhnout toho, že si vaše stránky budou moci prohlédnout i lidé, kteří stále používají starší webové prohlížeče. Pro všechny návštěvníky vašich stránek tak bude daná informace přístupná bez ohledu na to, jakým prohlížečem si ji zobrazí.
4. ***Oddělení obsahu a vzhledu.***
   Dodržením standardů můžete upravovat, či dokonce měnit obsah nebo formu (vzhled) nezáfisle na sobě. Oba celky se navzájem neovlivní
5. ***Tvorba proměnlivých stránek.***
   Pomocí standardů je mnohem jednodušší vytvářet stránky s proměnlivým obsahem. Snadno lze tedy například vytvořit stránky s mnoha položkami seznamu či položkami nabídek pro virtuální obchodní dům.
6. ***Kontrola kódu je vhodná.***
   Během návrhu webové stránky můžete využít služeb kontroly XHTML i CSS. Tato služba vám pomůže rychle odhalit případné chyby. Jakmile projdou dokončené stránky i touto kontrolou, budete si jisti, že jim nic nechybí.
7. ***Plynulost tvorby.***
   Tvorba pomocí standardů je mnohem efektivnější. Bez jejich uplatnění by byl webový návrhář jediný, který se dokáže vyznat mezi jednotlivými značkami a identifikovat tak pravý obsah

dokumentu. Dodržováním standardů můžete práci rozdělit mezi několik lidí, kteří pouze dodržují dohodnutá pravidla. Práce vývojářů se pak nakonec sama spojí s obsahem a vytvoří prezentaci.

8. ***Snadná distribuce obsahu.***
   Distribuce obsahu do aplikací jiných společností je mnohem jednodušší, jelikož je obsah oddělen od pravidel formátování, jejichž použití by stejně nebylo vhodné.
9. ***Zvýšení přístupnosti.***
   Díky dodržování standardů budou vaše stránky přístupnější pro uživatele se zrakovým postižením. V Americe musí státní instituce své webové stránky vytvářet v souladu se záknem, který je též znám jako ADA 508.
10. ***Usnadněte si práci.***
    Vytvoříte podstatně méně kódu v kratším čase. Výsledná práce bude lépe odpovídat požadavkům a bude jednodušší jí následně upravovat.

[Wyke-Smith, Charles: CSS, Využijte kaskádové styly naplno. 1, Brno, Computer Press a.s. 2006 ,s.13.]

1.2.1.1 XHTML

Zkratka XHTML (eXtensible HyperText Markup Language) znamená „rozšiřitelný hypertextový značkovací jazyk“. Definuje **strukturu** dokumentu. Protínají se zde vlastnosti jazyka XML a HTML. Nejvýraznější změna je že dokument musí být správně formovaný a že všechny elementy musejí být ukončené i ty nepárové, na to se použije zkrácený ukončovací tag. Dále jsou všechna jména elementů a atributů malými písmeny.

1.2.1.2 CSS

Zkratka CSS (Cascading Style Sheets) znamená „Kaskádové styly“. Definuje **formu** daného dokumentu, strukturu ovlivnit nemůže, tím je dosaženo oddělení formy od struktury. Tímto se dosáhne jednoduchost případné změny prezentace dokumentu. CSS slouží k ovládání formátování a umístění prvků stránek.

1.2.1.3 SEO

SEO (Search Engine Optimization), jedná se o optimalizaci pro vyhledávače, např. Google aj., čím lépe optimalizovaný web bude, tím lépe bude vyhledávač vyhodnocovat shodu při hledání nějakých klíčových slov.

Je potřeba dodržovat tyto základní pravidla:

1. **Titulek stránky**
   Vyhledávače často indexují slova z titulky stránky. Je tedy důležité vyplňovat <title></title> v hlavičce dokumentu, tak aby obsahoval důležitá klíčová slova.

2. **Nadpisy**

   Jedná se tedy o nadpisy <h1> až <h6>. Nadpis nejvyšší úrovně by pak měl obsahovat stejný text jako v titulku a smí se vyskytovat v dokumentu pouze jednou, většina vyhledávačů totiž porušení tohoto penalizuje.

3. **Odkazy a obrázky**

   Do hypertextových odkazů vždy umístíme nějaký logický obsah. Vyhledávač navíc lépe indexuje odkazy, které mají vyplněn atribut title, protože počet klíčových slov tímto vzroste.
   U obrázků zase pomáhá atribut alt. Oba tyto atributy jsou povinné aby byl kód validní.
   Dále by neměla stránka obsahovat odkaz sama na sebe, některé vyhledávače totiž za takovéto smyčky penalizují.

4. **Zvýraznění textu**

   V XHTML se jedná o tagy a , výrazy uzavřené do těchto tagů pak vyhledávač indexuje s vyšší prioritou než normální text.

5. **Hustota klíčových slov**

   Vyhledávač indexuje slova z textu, které se v něm nejčastěji opakují. Ovšem na toto neexistuje přesné pravidlo, které by určovalo kolik klíčových slov psát do textu. Nejlépe je se řídit vlastním úsudkem a slova používat tak aby nebyla samoúčelná.

6. **Adresy jednotlivých stránek webu**

   Naprosto nevhodný je nic neříkající PHP zápis s parametry. Např.
   http://www.mojestranky.cz/?fce=show&i=10
   Adresa by se měla skládat z klíčových slov, které stránku nejlépe vystihují a tyto slova oddělovat znaky _ nebo -, přičemž znak pomlčka má ve vyhledávačích výhodu.
   Pokud se stránky řadí navíc do kategorií, například u některého internetového obchodu, nejvhodnější tvar adresy je potom:
   http://mujeshop.cz/potraviny/bio/bio-brambory

7. **Výměna odkazů s ostatními stránkami**

   Pokud umístíme odkaz na naše stránky na stránky jiné, pomáhá to při indexaci našeho webu. Ovšem nejlepší je se do vyhledávačů registrovat manuálně.

8. **Aktualizace obsahu**

   Pokud se obsah pravidelně aktualizuje, roboti kteří web procházejí zaznamenávají změnu obsahu a znovu indexují stránky, toto může pomoci k lepší pozici při vyhledávání.

1.2.2 Prototype

Z hlediska funkčnosti algoritmů v prohlížečích IE, Opera, Firefox a Safari, jsem se rozhodl použít volně dostupný framework Prototype viz. http://prototypejs.org/.

Přináší velké usnadnění programování v javascriptu, zjednodušuje používání AJAX, vkládání do DOM, procházení DOM. Pro demonstraci uvedu některé nejpoužívanější funkce prototype.

1.2.2.1 Pomocné metody

Základní metody pro získání DOM uzlů. Jsou základním kamenem pro efektivní programovaní v Prototype. Nejpoužívanější je metoda **$**, která přidá takto získanému uzlu další funkcionalitu. Tyto metody jsou tak běžně používané, že jejich zápis byl uzpůsoben na jednoduchou a krátkou syntaxi.

1.2.2.2 Metody elementu

Jedná se tedy buď o získání daného elementu, nebo pole elementů a práci s nimi. Elementy takto získané jsou opět extended, což dovoluje jejich řetězení.
Např.: $(element).down(1).next('li', 2).hide();
pozn. Všechny tyto funkce ignorují textové uzly, vrací pouze elementy.

ancestors

element.ancestors() -> [HTMLElement...]
Všechny předky daného elementu vrátí jako pole extended elementů. Vrací včetně uzlů <body> a <html> viz. obrázek 1.1

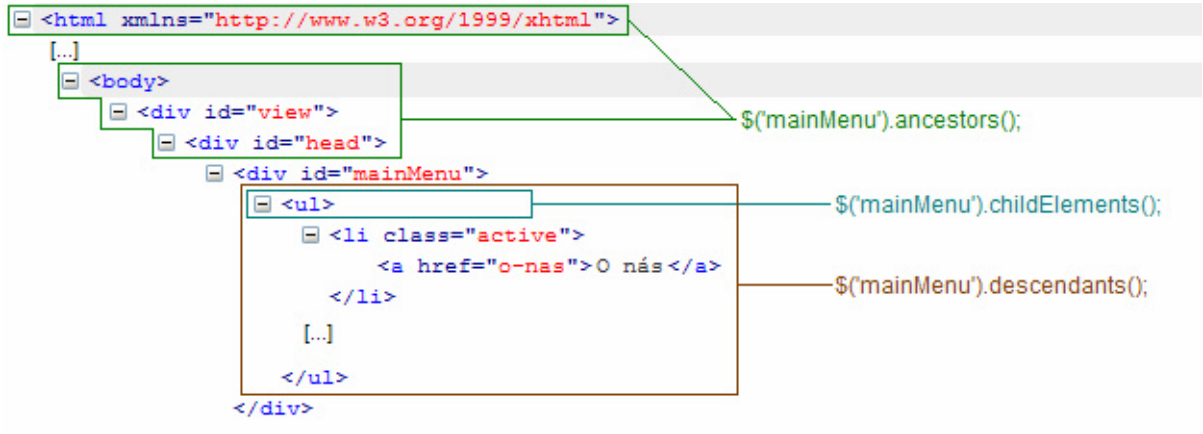
childElements

element.childElements -> [HTMLElement...]
Všechny bezprostřední potomky daného elementu vrátí jako pole extended elementů viz. obrázek 1.1

descendants

element.descendants() -> [HTMLElement...]
Narozdíl od funkce childElements, descendants vrací všechny potomky ne pouze bezprostřední viz. obrázek 1.1

Obrázek 1.1 ilustrující metody ancestors, childElements a descendants. Vygenerováno programem Firebug.



down

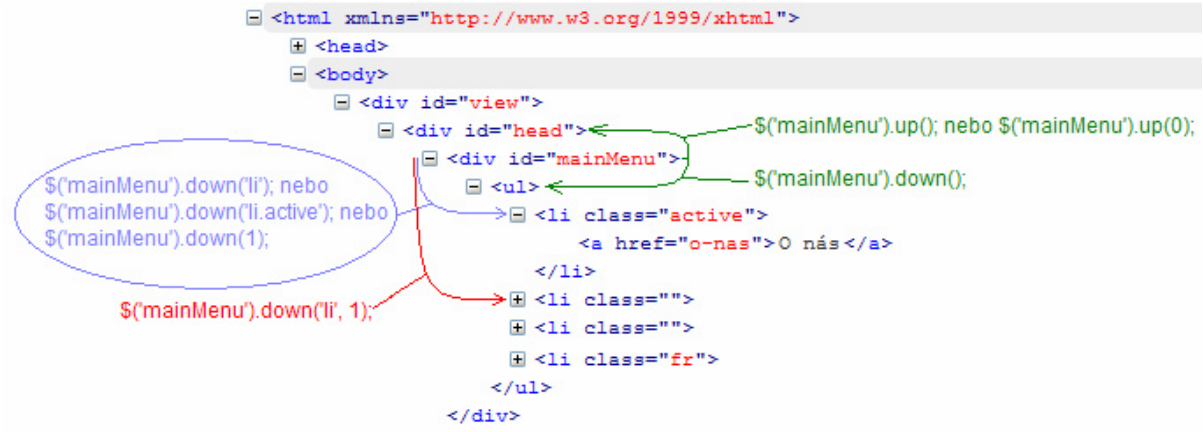
element.down([, cssRule][, index = 0]) -> HTMLElement | undefined

Vrátí prvního, nebo n-tého potomka, který splňuje css pravidlo. Pokud pravidlo není zadáno, uvažují se všichni potomci. Vrací undefined pokud nenalezne shodu. Hledání pomocí indexu je stejné jako indexování pole descendants, v tomto jednorozměrném poli všech následníků se vybere n-tý prvek, indexuje se od 0. Viz obrázek 1.2.

up

Stejně jako **down**, jen uvažujeme předky elementu. Viz obrázek 1.2.

Obrázek 1.2 ilustrující metody up a down. Vygenerováno programem Firebug.



next

element.next([, cssRule][, index = 0]) -> HTMLElement | undefined

Vrací následující, nebo n-tý uzel, který splňuje css pravidlo. Pokud pravidlo není zadáno, uvažují se všichni potomci. Vrací undefined pokud nenalezne shodu. Viz obrázek 1.3.

nextSiblings

element.nextSiblings() -> [HTMLElement...]

Všechny následující elementy vrátí jako pole extended elementů. Viz obrázek 1.3.

previous, previousSiblings

Obdobně jako **next, nextSiblings**, uvažujeme předchozí uzly. Viz obrázek 1.3.

Obrázek 1.3 ilustrující metody next, nextSiblings, previous a previousSiblings . Dále ukázka zřetězení metod. Vygenerováno programem Firebug.

1.2.2.3 Metody pro práci s polem elementů

Metody uvedené v předchozí kapitole často vrací pole elementů, které lze metodami níže uvedenými snadno zpracovávat nebo procházet pomocí iterátoru.

Pro práci s polem extended se používá iterátor **each**. Poté se používají funkce pro práci s jednotlivými elementy. Může se uvádět parametr index, což je index elementu v poli.

Př.:

Iterátorem projdeme všechny prvky pole, které vrací metoda nextSiblings a postupně je smažeme z DOM.

```
$(element).nextSiblings().each(function(element, index)
{
	element.remove();
});
```

1.2.2.4 Vkládání

Vkládání HTML kódu na zadané umístění. Pokud html obsahuje JScript kód uzavřený do <script> tagu, tento kód bude vyhodnocen.
Pomocí klíčových slov **after**, **before** se vloží html jako následující nebo předchozí uzel elementu. A pomocí **top**, **bottom** vloží html jako prvního nebo posledního potomka elementu.

1.2.2.5 Formuláře

Funkce pro práci s formuláři, jelikož při práci s AJAX nelze odesílat formuláře klasickým způsobem, jsou zavedeny metody pro serializaci dat, popřípadě se simuluje HTTP metoda odeslání. Pokud ovšem formulář obsahuje soubory, je nutné použít iframe.

serialize

form.serialize() -> String

Serializuje formulářová data do podoby vhodné pro poslání pres AJAX.

request

request([options])

př.:

```
$('form').request(
{
	method: 'get',
	onComplete: function(transport)
	{
		alert(transport.responseText);
	}
});
```

Někdy nestačí pouze serializovat formulář, ale je nutné přímo přepsat HTTP metodu odesílání formuláře, aby byl dosažen požadovaný efekt. Například při použití WYSIWYG editoru atd..

1.2.3 AJAX

Asynchronní JavaScript a XML. Navzdory jménu není povinné použít ani jedno z nich. Vznik přibližně v roce 1999, byl vytvořen **XMLHttpRequest** object pro Internet Explorer 5, ostatní prohlížeče následovaly tohoto příkladu.
Využití AJAX výrazně zrychluje práci uživatelům, přenášené množství dat je mnohem menší a není potřeba načítat znovu celou stránku. Například načítání JScript frameworků tedy proběhne jen

jednou, čímž vzniká velká časová úspora a systém je svižnější. Také některé operace lze provádět pouze na pozadí, na popředí pak uživatel vidí pouze operace prováděné javascriptem. Asynchronnost lze efektivně využít například při obnovení různých nezávislých částí webového systému.

Prototype poskytuje jednoduchou tvorbu AJAX požadavku:

Ajax.Request

```
new Ajax.Request(url[, options])
př.:
var url = '/www.someurl.com';
var form =  $('someform');

new Ajax.Request(url,
{
        parameters: form.serialize(),
        method: 'post',
        onSuccess: function(transport)
        {
                alert(transport.responseText)
        }
});
```

Celkové možnosti stavby ajax požadavku na http://prototypejs.org/api/ajax/options. Callback funkce nejsou podporovány ve všech prohlížečích.

Za zmínku stojí také **Ajax.PeriodicalUpdater**, vytvořením tohoto objektu, se začne opakovaně provádět AJAX požadavek, který byl zadán (stejná syntax jako klasický AJAX požadavek). Ukončuje se zavoláním metody stop, znovu spouští start metodou. Lze zadat frekvenci požadavků a součinitel útlumu (decay), pokud je od serveru stejná odpověď, perioda požadavků se zvětší vynásobením tímto součinitelem (tedy součinitel musí být větší než 1), stejně tak se zvětšuje i součinitel. Až se odpověď změní, vše se nastaví na původní hodnoty. Touto cestou se dá například jednoduše udělat diskusní fórum, které se obnovuje přes AJAX, aj.

1.2.4 Scriptaculous

Scriptaculous je knihovna pro pohodlnou práci s grafikou a drag & drop objekty.
Z hlediska funkčnosti algoritmů v prohlížečích IE, Opera, Firefox a Safari, jsem se rozhodl použít volně dostupný framework script.aculo.us viz. http://script.aculo.us/ postavený na Prototype viz. http://prototypejs.org/.

1.2.4.1 Animation framework

Obsahuje mnoho funkcí pro grafické efekty. Jako zpomalené objevování, mizení elementu, posouvání na určitou pozici, zvětšování, zmenšování elementů. Pomocí těchto nástrojů lze vytvořit vizuálně přitažlivý systém. Operace jsou poměrně náročné na operační paměť, proto jsem se rozhodl je používat střídmě vzhledem k možnostem uživatelů.

1.2.4.2 Drag and drop

Drag and drop objekty jsou poměrně intuitivní, zrychlují práci například při posunování položek v seznamu, nebo matici elementů.

Vytvoření draggable objektu :

new Draggable('id_of_some_element', [options]);

Objekt nyní reaguje na onmousedown událost nad tímto elementem. Pomocí options lze nastavit mnoho druhů chování.

např.:

revert: true,false – element se vrátí na své místo při události onmouseup, toto chování se přetíží funkcí onDrop, pokud je objekt umístěn do správné drop zóny.

constraint: horizontal, vertical – lze omezit pohyb elementu jen v jednom směru

starteffect, *reverteffect*, *endeffect* – lze jednoduše přidat změnu CSS třídy elementu při těchto událostech draggable objektu.

Vytvoření droppable zóny:

Droppables.add('id_of_some_element',[options]);

Objekt reaguje na onmouseup událost nad tímto elementem. Spouští se callback funkce onDrop definovaná programátorem.

Důležitý parametr options je *accept*, u droppables lze pomocí css třídy zvolit, který draggable objekt může přijmout.

Při využití AJAX je důležité mazat droppables předtím než je smazán uzel z DOM. A to pomocí Droppables.remove(element); příkazu. Ponechání reference by způsobilo nefunkčnost drag and drop.

Stejně tak důležité je mazat draggable objekty, zavoláním metody destroy(); . Při ponechání je sice vše funkční, ale garbage collector nemůže objekt odstranit a operační paměť se zahlcuje.

1.2.4.3 Sortable

Obsahuje v sobě jak vytvoření draggable tak i droppables objektů.

Jedním příkazem se vše inicializuje Sortable.create('id_of_some_container',[options]);

Zrušení pak pomocí Sortable.destroy('id_of_some_container');

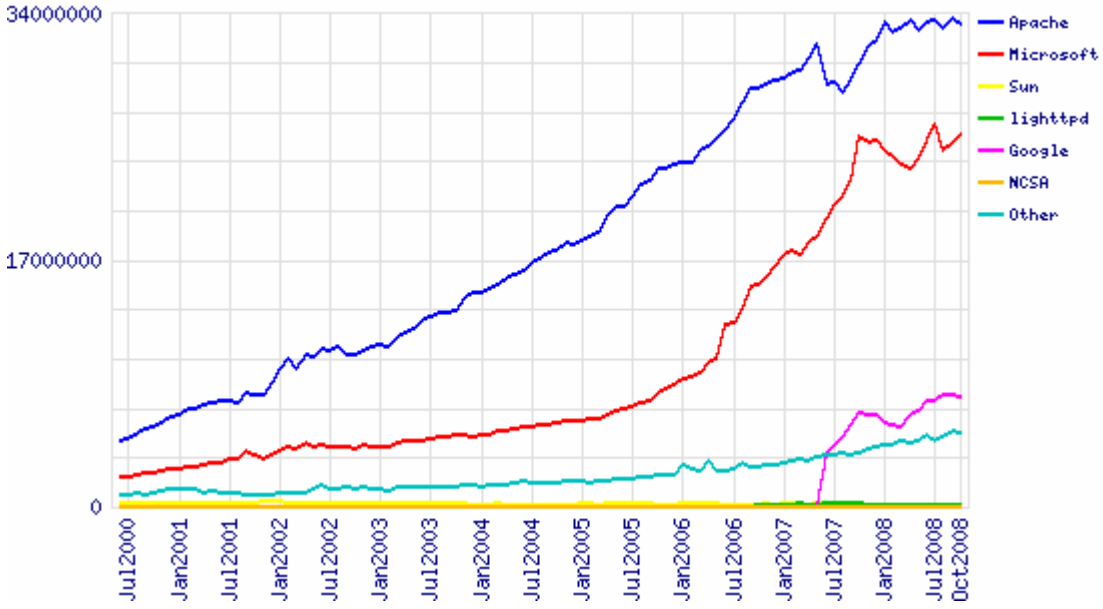
Jde tedy o rychlé vytvoření drag and drop stromu, nejčastěji z HTML seznamu. Pozice jednotlivých elementů se zjistí příkazem serialize(), a následně se odešle na server, tedy změna pozice probíhá na popředí i na pozadí. Odesílají se ID elementů v poli, které má stejné pořadí jako sortable strom.

1.2.5 PHP

PHP je nejrozšířenější webová platforma na světě. Je oficiálním modulem Apache HTTP serveru. Pokud je PHP skriptovací engine takto vestavěn do serveru samotného, znamená to rychlejší zpracovávání, efektivnější alokaci paměti a také snadnější údržbu. Tedy nejčastější použití PHP je spolu s Apache serverem a MySQL databází. Tato kombinace je prozatím také nejlevnější řešení při zřizování webového portálu. Pro ilustraci uveden celkový počet Apache serverů a užívání PHP.

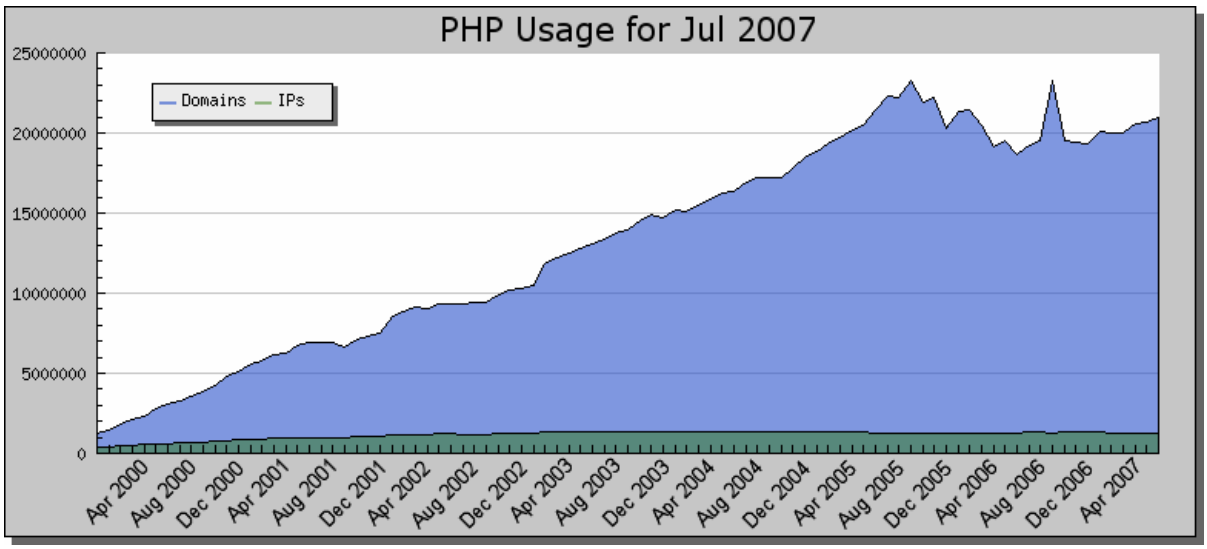
Obrázek 1.4: Celkový počet aktivních serverů přes všechny domény:

Zdroj: Netcraft



Obrázek 1.5: Graf užívání PHP:

Zdroj: Netcraft



PHP je zkratkou pro „Hypertext Preprocesor“, původně „Personal Home Page Tools“. Nejnovější verzí PHP je PHP5.

Je to skriptovací programovací jazyk běžící na straně serveru, určený především pro tvorbu webových stránek. Může být přímo vložen do HTML kódu, nebo být spouštěn samostatně (toto řešení se používá nejčastěji v kombinaci se šablonovým systémem). Výsledkem PHP kódu bývá většinou HTML kód. Je to jazyk interpretovaný, neprobíhá tedy žádná kompilace kódu, ačkoliv existuje Zend encoder, který převede kód do předkompilované formy. Toto se používá jak pro zrychlení interpretace kódu, tak pro zatemnění kódu.

PHP obsahuje mnoho rozšíření pro komunikaci s programy a protokoly. Velká podpora databázových rozhraní (MySQL, Oracle atd.). Podpora protokolů jako POP3, IMAP, LDAP a mnoho dalších.

Dále je zabudována GD grafická knihovna, podpora XML pomocí DOM, SimpleXML, XSLT atd.

PHP je jazyk dynamicky typovaný, nepotřebuje deklaraci proměnných předtím než je do proměnné přiřazena hodnota. Při použití proměnné, do které nebylo přiřazeno, má proměnná defaultní hodnotu podle kontextu, kde byla použita. Tedy pokud je očekáváno číslo má hodnotu 0, pokud řetězec má hodnotu prázdného řetězce.

1.2.5.1 Třídy a objekty

PHP verze 5 odpovídá na požadavek na více objektově orientovaných prvků v PHP. Byla přepsána objektově orientovaná část Zend enginu, ačkoliv jmenné prostory a vícenásobná dědičnost, zmíněné v dokumentu „Zend Engine II: Feature Overview and Design“, implementovány nebyly. Vícenásobnou dědičnost lze implementovat pomocí Interface.

PHP5 začíná také pracovat s ukazateli na objekty, v předchozích verzích se při přiřazení proměnné, která ukazovala na objekt, vytvářela nová kopie objektu. V PHP5 už toto neplatí, kopie se vytváří klíčovým slovem **clone**, jinak se pracuje pouze s ukazateli. Už tedy není nutné předávat objekt pomocí reference nebo přiřazovat referenci na objekt.

Přidává public/protected/private modifikátory přístupnosti metod a atributů.

Poté přidána funkce instanceOf(), pro zjištění zdali je objekt instancí dané třídy. Další přidané prvky jsou popsány podrobněji níže.

Constructors and Destructors:

function __construct() {}

Volá se při vytváření instance třídy.

function __destruct() {}

Destruktor se zavolá sám při dokončení PHP skriptu.

extends:

class Child_class extends Main_class {}

Pouze jednoduchá dědičnost, vícenásobná není podporována, záleží na pořadí definice tříd, první musí být definovaná třída, ze které se dědí.

interface:

Jak bylo řečeno, třída může dědit pouze z jedné třídy, ale zato může implementovat libovolný počet Interface.

```
interface Show{
        function show();
}
interface Save{
        function save();
}
interface Item extends Save,Show{} // zde je tedy vícenásobná interface dědičnost
```

```
class ItemClass implements Item{
        // tady musí být implementovány všechny metody, které jejichž deklarace byla
        // zděděna, v tomto případě tedy metoda Show a Save
}
```

Final methods, classes:

Klíčové slovo **final** u metod říká, že tato metoda nemůže být přetížena metodou ze zděděné třídy. U tříd znamená, že tato třída nemůže být zděděna.

Static methods, members:

Statické metody jdou volat samostatně, nemají žádnou vazbu na objekt. Statický atribut je potom přístupný přes třídu, nevztahuje se k objektu.

1.2.5.2 Funkce pro práci s databází

Obsahuje rozsáhlou sadu funkcí pro práci s databází, počínaje připojením k databázi, selekcí databáze. Dále funkci pro provedení dotazu mysql_query(), která vrací identifikátor výsledku dotazu. Tento se pak použije jako parametr některé z funkcí pro získání řádků z databáze, např. mysql_fetch_array() nebo mysql_result(), nebo pouze pro test zdali se dotaz provedl v pořádku, pokud jsme provedly update, vkládání nebo jinou operaci s tabulkou či databází.

1.2.5.3 Spravování sezení

Protože **protokol HTTP je bezstavový**, je potřeba mít způsob jak dodat kontext požadavku HTTP. Tedy informace o předchozích či následujících požadavcích. Řešení tohoto problému bylo zpočátku pouze cookies, které se ukládá na klientovi. Ovšem omezená velikost a počet cookie dali vzniknout Session.

Při používání Session je každému klientovi přiřazeno session ID (SID). Pomocí tohoto ID se poté dostaneme ke všem potřebným údajům o uživateli. Aby bylo možné přistupovat k session, klient a server musí mít někde uloženo jaké SID je spolu svazuje. Nejčastější řešení je uložení SID do Cookies. Tím, že se uloží pouze ID místo všech dat, je vyřešen problém s omezenou velikostí cookies. Klient ovšem musí mít cookies povoleno. Jinou možností je přidávat SID natvrdo do každé URL, toto řešení ovšem není bezpečné, více klientů by tak mohlo používat stejný SID. Pomocí PHP pak základní zacházení s Session probíhá jednoduše. Pomocí funkce **session_start()** se vytvoří nové sezení nebo se pokračuje v sezení aktuálním. Poté už lze používat tuto super globální proměnnou **$_SESSION**. Lze do ní zapisovat jako do klasického pole, jednotlivé prvky se pak ruší pomocí funkce **unset()**.

Jako příklady použití session je například rychlé přihlášení, tedy uložení informací o přihlášeném uživateli. Dále je pak ostatní informace o uživateli, které se neukládají do databáze, nebo se nemohou ukládat do databáze, protože uživatel nemá vytvořený účet. Pro všechny takovéto dočasné informace je pak session nenahraditelné.

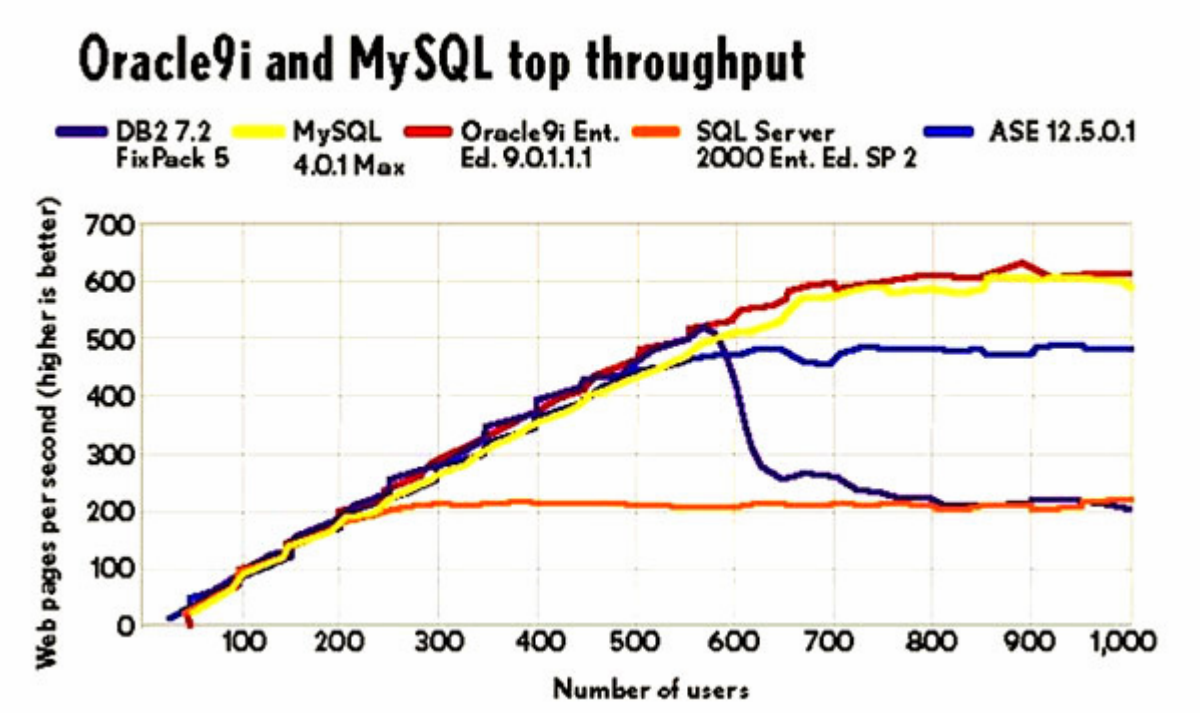
1.2.6 MYSQL

Při vytváření webového portálu je důležité ukládat data tak, aby k nim šlo **snadno přistupovat a to podle nejrůznějších kriterií**. Tedy je možné i ukládání do souborů v různých formátech, např. XML či XLS, ale používat toto při běžné práci s webem by bylo nevhodné. Tyto formáty se tedy použijí spíše pro import či export do databáze, lze tak jednoduše upravovat například seznam produktů a následně ho importovat.

Pro webový portál vhodné použít databázi a to z důvodu efektivního ukládání dat, vyhledávání, třídění a náhodného přístupu k datům. Databázový server dále zajišťuje, že více uživatelů může s daty pracovat zároveň a autorizovaným uživatelům poskytuje rychlý přístup k datům.
MYSQL je velmi rychlý, robustní relační databázový systém. MYSQL je více-uživatelský, více-vláknový server. Používá jazyk SQL (Structured Query Language).

Od začátku tvorby MySQL byly kladeny hlavní požadavky na rychlost produktu a jeho rozšiřitelnost. Ačkoliv ze začátku postrádal schopnosti jako uložené procedury, triggery nebo transakce, přitáhl k sobě tento produkt velké množství klientů.

Obrázek 1.6: Porovnání rychlostí databázových serverů:
Zdroj: eWeek



1.2.6.1 Specifické optimalizační schopnosti

Kromě optimalizované základny MySQL, stojí za zmínku i některé specifické optimalizační schopnosti MySQL.

Replikace

Replikace dat umožňuje jejich duplikaci na jeden či více serverů. Tím se rozprostře zátěž na více serverů, což přinese výrazné zrychlení, nemluvě o větší bezpečnosti, protože jsou data zálohována na více serverech a také možnosti výpadku některých serverů bez omezení funkčnosti aplikace, která se na server nachází.

Cachování dotazů

„Cachované dotazy, to je jeden z největších příspěvků MySQL pro zvyšování výkonu. Když je tato schopnost zapnutá, funguje jednoduše a velmi efektivně. MySQL si pak totiž uchovává dotazy SELECT spolu s jejich výsledky. Když se spouštějí následné dotazy, porovnává je MySQL s cachovanými dotazy, jestliže jsou shodné, MySQL obejde nákladné získávání dat z databáze, a prostě vytáhne výsledky cachovaného dotazu. Aby se zamezilo zastaralým výsledkům, jsou zabudované i mechanizmy, které automaticky odstraňují zastaralé cachované výsledky, a nahrazují je při příštím požadavku novými.“

[Gilmore, W. Jason: Velká kniha PHP 5 a MySQL, kompendium znalostí pro začátečníky i profesionály. 1, Brno, ZONER software s.r.o. 2005 , s. 504.]

Lokalizace

K dispozici j více než 30 znakových sad používaných v součinnosti s více než 20 jazyky.

Konfigurační volby

Lze povolit pouze určitý počet připojení, aktualizací atd. za hodinu. Nastavovat práva uživatelům. Podpora SSL certifikátů.

1.2.7 SMARTY

Při tvorbě větších projektů se bez šablonového systému nelze obejít, oddělení logické a prezentační části je velice důležité, protože při změně jedné části zůstává další neovlivněna. Prezentační část vytváří většinou jiný člověk než PHP programátor, zajišťuje se pouze propojení šablony a PHP skriptu. Program v PHP je pak nezávislý na použité prezentaci, tedy lze ho napojit na více grafických

výstupů. Navíc je takto psaný kód mnohem přehlednější. Práce s šablonami umožňuje více možností, šablony se mohou vkládat na více míst, lze jednoduše iterovat části šablony, provádět podmíněné výpisy aj.

Preparser pak převede šablony do PHP kódu a spuštění tak dosáhne stejné rychlosti jako by šablonový systém nebyl použit. Pokud byla šablona změněna, Smarty její obsah automaticky překompiluje.

1.2.7.1 Výhody SMARTY

Pomocí PHP většinou stačí pouze vypsat data z databáze a nemusí se k nim přidávat informace, které by se projevovaly pouze v grafickém výpisu, což výrazně zpřehledňuje výsledný PHP kód. Tedy informace jako počet vypsaných záznamů, pořadí položky ve výpise atd. se do šablony přidávají automaticky.

Dále SMARTY podporuje veškeré matematické operace. Také textové operace, což jsou funkce nazývající se modifikátory proměnných, např. oříznutí textu, zalamování textu, nahrazování textu a to i pomocí regulárních výrazů, změnu formátu data atd.

Další důležité schopnosti smarty:

- ***Cachování šablon (caching).***
  Smarty také nabízí volitelnou možnost skladovat šablony v souborech. Liší se to od kompilace v tom, že zapnete-li cachování, zabráníte dokonce tomu, aby se daná logika vykonávala; realizuje se jen uskladněný obsah. Například, můžete vyznačit dobu života uskladněných dokumentů, řekněme na pět minut. Během této doby se tak vyhnete databázovým dotazům vztahujícím se k šabloně.
- ***Vysoce konfigurovatelný a rozšiřitelný.***
  Objektově orientovaná architektura Smarty umožňuje modifikovat a rozšiřovat jeho výchozí chování. Kromě toho, od úplného počátku bylo návrhářským cílem, aby byl dobře konfigurovatelný, takže se uživatelům nabízí značná flexibilita, co se týče přizpůsobování chování Smarty. Děje se to prostřednictvím zabudovaných metod a atributů.
- ***Zabezpečení.***
  Smarty nabízí řadu schopností, jejichž účelem je odstínit data serveru i data aplikace před potenciálním poškozením ze strany designéra, ať už záměrným nebo jiným.

[Gilmore, W. Jason: Velká kniha PHP 5 a MySQL, kompendium znalostí pro začátečníky i profesionály. 1, Brno, ZONER software s.r.o. 2005 , s. 408.]

1.2.7.2 Nejpoužívanější funkce v šablonách

{if},{elseif},{else}

Podmíněný výpis, podmínka může obsahovat libovolný výraz.

{foreach},{foreachelse}

Iterace jednorozměrného pole, jde přistupovat jak ke klíči tak hodnotě.

{section},{sectionelse}

Iterace přes dvourozměrné pole, iteruje se podle indexu, jde nastavovat krok iterace. Section lze navzájem vnořovat.

1.2.7.3 Nejpoužívanější funkce v PHP

Musí proběhnout vytvoření smarty objektu
$smarty = new Smarty();

Poté probíhá např. nastavení cest k adresářům se šablonami, povolení cachování a jiné parametry smarty objektu.

Pokud se celková šablona skládá z několika šablon, je dobré si tento objekt předávat. Všechny data se do něj načtou pouze jednou a tím se omezí přístup do databáze.

assign

Přiřazení PHP mixed proměnné do šablony. Např. vícerozměrného pole, což může být několik záznamů z databáze.

display

Zadává se parametr název šablony, provede se přiřazení proměnných a šablona se zobrazí

fetch

Vrátí obsah šablony, místo jejího zobrazení. Takto jde například skládat struktura pomocí rekurze, tedy například obecné menu s více úrovněmi. A skládat celkovou šablonu z několika dílčích šablon.

2 Cíl práce

Tvorba a modifikace položek v systému. Strukturu a obsah položky lze měnit přímo ve webovém rozhraní pomocí přehledného průvodce vytvořením šablony. Struktura se mění pomocí drag and drop. Celá tvorba pouze pomocí javascriptu, při uložení se vytvoří šablona a odpovídající struktura v databázi.

Položka půjde vytvořit z připravených pluginů. např.

- foto galerie, video galerie, galerie souborů
- obecné formulářové prvky
- kalendář
- WYSIWYG editor
- diskusní fórum

Umístění těchto položek kamkoliv do hlavního stromu. Poté vyřešit obecně možnosti řazení těchto položek v kategoriích, v závislosti na stejném obsahu různých položek. Nastavení defaultního řazení. Což se bude projevovat i ve frontendu prezentace.

Pro zrychlení práce přepínání mezi základními panely:

- detail

  Zde bude výpis naposledy otevřeného detailu položky.

- kategorie

  Výpis naposledy otevřené kategorie, libovolné řazení, limit zobrazení, stránkování. Dále ve vývoji množstevní editace vybraných položek a vybraných sloupečků z DB.

- vyhledávání

  Výpis posledního vyhledávání, označení kontextu vyhledání.

Položky lze vytvářet pouze v určitých jazycích, tedy lze dané jazykové mutace mazat či znovu vytvářet.

Obecně rozlišovat přidání jazykového (různá textová data) a nejazykového pluginu (datum, váha cena). Např. položka je ve třech jazycích, obsahuje galerii, která je všech jazycích stejná, ale jazyková data (alt, title jednotlivých obrázků) má ve všech jazycích, a pak galerii, která je v každé jazykové mutaci jiná.

Fulltextové vyhledávání v celém stromu, popřípadě v dalších částech systému.

Základní správu uživatelů. Zatím vyřešit pouze základní práva, administrátor a práva pro nás tvůrce webu. Později podle požadavků klientů vyřešit práva pro zobrazení, editaci, mazání, přidávání. A to buď vzhledem k typu položky, nebo vzhledem k umístění položky ve stromu.

3 Řešení

Použití AJAX sebou rovněž nese celou řadu nevýhod. Zejména nemožnost listovat historií a předávat si odkazy na určité části systému, např. články, pomocí url. Toto lze částečně obejít tím, že bude poslední otevřená kategorie zachována na pozadí a přepínat mezi kategorií a detailem půjde pomocí panelů. Dále by bylo vhodné ukládat stav systému do session, tedy stav obsahové části, stav menu a stav jednotlivých pluginů, aby bylo možno po obnovení stránky tento stav znovu načíst. Prohlížeče musejí mít zapnutou podporu JScript a musejí AJAX podporovat. Výhody jsou svižnější systém, méně přenesených dat, možnost provádět operace na pozadí.

Motivace

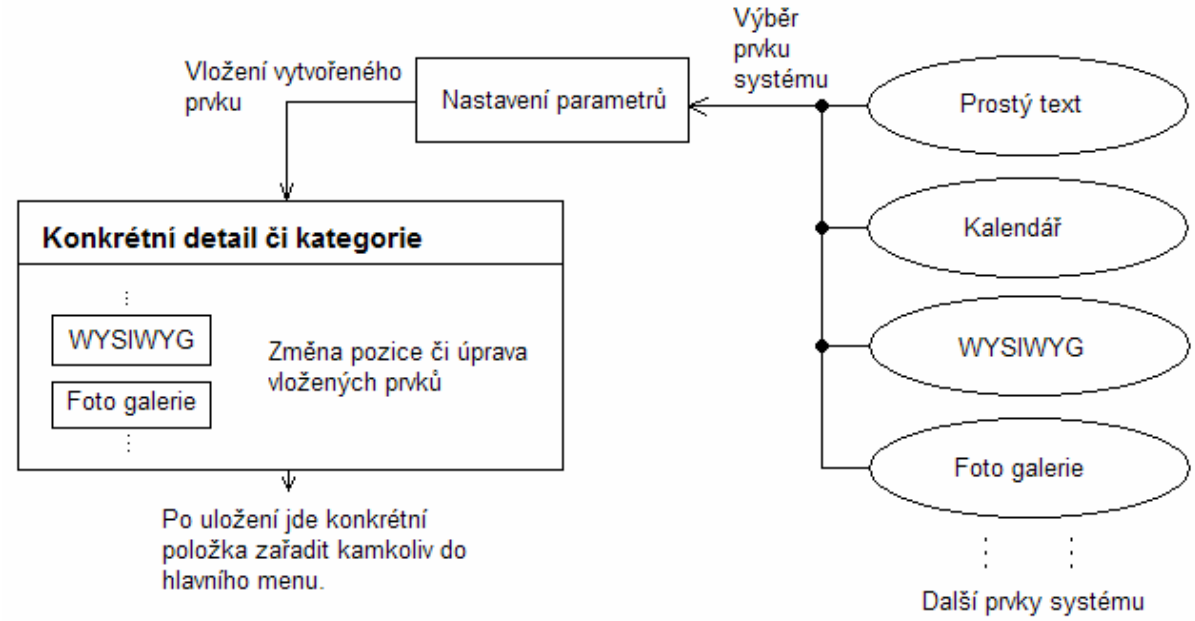
Tento systém jsem se rozhodl vytvořit ve snaze zautomatizovat některé opakující se výkony při tvorbě prezentací nebo internetových obchodů. Přičemž i internetový obchod obsahuje prezentační část, tedy nějaké informační stránky, kromě samotných produktů. Zkusil jsem k tomuto problému přistoupit co nejvíc obecně. Tedy chci mít obecný strom, kam mohu zařadit položku jakéhokoliv typu, chci aby se tyto položky dali libovolně řadit a strukturovat. K tomu potřebuji dva základní typy položky. Kategorii, která dokáže obsáhnout další položky a detail, který může být libovolnou položkou. Libovolná položka znamená, že může být třeba produktem, nebo nějakou stálou stránkou. Ale například i seznam uživatelů, kteří se na stránkách registrovali, takovýmto položkám pak jen stačí nastavit aby se nezobrazovali na webu, uživatel webu, tak bude moci zobrazit maximálně sám sebe. Takto musí být možné zde strukturovaně uložit celou webovou část. ***Hlavní motivací tedy bylo vytvořit datový sklad, kde se s položkami manipuluje nezávisle na jejich konkrétním typu, ale pouze podle toho jestli se jedná o detail nebo kategorii.***

Vytvoření konkrétního položky (jedná se o typ ne instanci)

Řekněme, že mám vytvořen obecný strom, kam mohu zařadit cokoliv co mě napadne. Krásná představa, ale zařazení položky manuálně do takového stromu by bylo příliš pracné, tedy pokud uvažuji, že musím nastavit relaci se stromem, tak aby byla položka unikátní, manuální řešení by nebylo příliš pracné. Zkusil jsem ale přemýšlet dál. Dejme tomu že položka má název a odkaz, ovšem tyto jsou různé ve všech jazycích, tedy musím je nějakým způsobem oddělit, nejlépe do více tabulek, s tím přichází zajištění dalších relací a to už v rámci položky samotné. Ovšem zde to nekončí, položka může obsahovat například foto galerii, ovšem i ta obsahuje jazykové a nejazykové texty, dokonce můžeme rozdělovat i na jazykovou a nejazykovou galerii. Zajišťování všech těchto relací a unikátnosti by bylo příliš pracné dokonce i pro mě tvůrce systému. ***Je zde tedy jasná motivace, vytvořit modul pro vytváření těchto položek, tak aby relace jazykových verzí a pluginů byly zajištěny automaticky.***

Má základní úvaha byla že, každá položka v systému se skládá z nějakých opakujících se elementů. Jsou to například formulářové prvky text, checkbox atd. a nebo větší prvky jako foto galerie, které už jsou zároveň autonomním systémem. Tedy chtělo by to nadefinovat tyto prvky obecně, tak aby je bylo možno vytvořit s uživatelskými parametry, jako například jméno nebo velikost textarey, či defaultní hodnota checkboxu. Tento obecný prvek se poté po vyplnění údajů stane konkrétním a lze ho vložit do libovolné položky. Jedná se tedy o vytváření typu položky, tím že do ní naskládám takovéto elementy, definuji jak bude výsledná položka vypadat, vznikne z ní v podstatě složitější struktura. Výhody definování si libovolného elementu, ze kterých lze poté vytvářet položky, lze rozvinout ještě dále, ovšem za předpokladu, **že si nějakým způsobem budu uchovávat jakého typu je můj element!** Zvolil jsem tedy syntax, kde je název typu a název konkrétního elementu oddělen znakem „_“, např. inputcheckbox_mujcheckbox. Uznávám že zvolená syntax je příliš dlouhá, proto další zásadní krok ve vývoji bude zkrácení těchto názvů typů. Tedy takto definované typy elementů, se stávají mým vlastním typem. Z tohoto mi vyplynula důležitá vlastnost. ***Při zpracování celkové položky (vytváření, či ukládání), je položka rozdělena na elementy, kde se každý automaticky zpracuje podle typu elementu, přičemž způsob zpracování každého typu je buď definován nebo se bere zpracování obecné.*** Tato vlastnost mi umožňuje definovat nové elementy a způsob zacházení s nimi, například element typu cena, bude se orientovat podle jazyků a k nim přiřazeným měnám, bude určovat, která měna je základní a zbytek si přepočítávat podle tabulky kurzů. Vše bude probíhat v rámci tohoto elementu, je tedy zapouzdřen a půjde přidat do libovolné položky velice jednoduše (velice jednoduše je ovšem relativní pojem, protože někteří klienti mají problém zamířit myší, je takováto tvorba položky pořád v rukou nás tvůrců výsledné aplikace). ***Ve shrnutí, pokud máme definován typ elementu(prvku) a způsob jeho obsloužení, je možno nastavit jeho parametry a přidat ho do konkrétní položky, viz obrázek 3.0.***

Obrázek 3.0. Vytvoření položky, ze které půjdou vytvářet instance



Takováto položka se pak uloží pod zadaným jménem a půjde přidat kamkoliv do hlavního menu. Například si takto vytvořím položku „novinka“, poté půjde v každé kategorii vytvořit instance novinka.

Vytvoření instance některé konkrétní položky

Tedy mám definovánu položku, pro lepší představu jí můžeme říkat opět novinka. Je načase si navrhnout strukturu. Řekněme že nám bude stačit jeden typ kategorie, nazveme ho obecná kategorie. Tato obecná kategorie je vytvořena stejně jako ostatní položky, viz obrázek 3.0, jen má typ kategorie. Teď si tedy mohu vytvořit instanci obecné kategorie, nazvu ji např. Novinky a v této kategorii už mohu vytvářet instance jednotlivých novinek. Důležité je, že v každé kategorii jde určit filtr pro vytváření položek a tento filtr se dále dědí či může být zděděn explicitně pomocí tlačítka zdědit. Je to proto, že uživatel nemusí mít zbytečně pravomoci přidávat všechno kamkoliv a za druhé by bylo matoucí, kdyby šly do kategorie Novinky přidávat například produkty. Dále lze nastavovat další atributy adresáře, jako podle jaké položky se bude řadit, zdali sestupně nebo vzestupně apod.

Propojení s výslednou prezentací, která je volně přístupná z internetu

Máme-li hotovou datovou základnu, tedy celou strukturu webu, např. články, aktuality apod., musíme ji nějakým způsobem propojit s výsledným webem. Tedy budeme provádět nad daty operace, nejčastěji se bude jednat o jejich zobrazení. Webová část je rozdělena na šablony podle typů dat, které bude obsahovat a tyto data budou přístupné pouze pomocí URL. Každá položka v systému má zároveň vygenerovaný i svůj unikátní link, podle názvu v dané jazykové mutaci a posloupnost takových odkazů pak ukazuje na danou položku, jedná se o posloupnost odkazů na kategorie, končící kategorií nebo detailem. Systém pak podle typu pozná, zdali se bude vypisovat seznam položek nebo daný detail a zároveň podle typu položky určí, do které šablony se bude vypisovat. ***Shrnu-li předcházející, vytvoří se hlavní šablona (layout) a vnitřní šablony, které se pojmenují podle typu položek, v těchto šablonách se vyplní proměnné v syntaxi Smarty na to místo, kde mají být příslušné výpisy a web je hotov.***

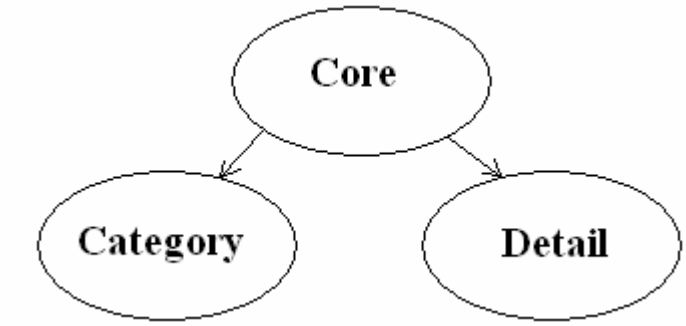
Probral jsem tedy logickou stavbu systému a posloupnost myšlenek, které mě vedly ke tvorbě v tomto tvaru. Snažil jsem se při návrhu řídit projekty, které jsem při své praxi implementoval, uvědomit si podobné části a tyto části navrhnout co nejvíc obecně. Dále už bych rád rozebral problém z programátorského hlediska, tedy jak navrhnout celý systém a na jaké entity ho rozdělit. Jednoduchý objektový model se zdá být vhodný.

Návrh systému pomocí objektového modelu

Při návrhu systému jsem se snažil navrhnout takový objektový model, aby šli hlavní objekty obsluhovat centrálně. Hlavní rozdělení je tedy na kategorii, což je ve své podstatě analogií adresáře v souborovém systému a detail, který je obdobou libovolného souboru. Tyto dvě hlavní entity

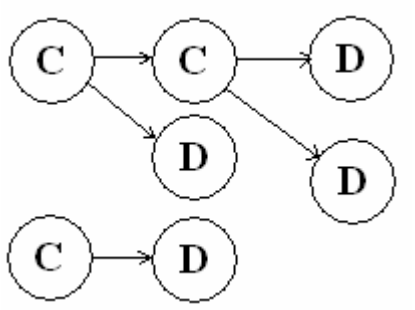
v systému poté mají i podobnou funkčnost, tedy budou mít svojí nadřazenou třídu, která tuto funkčnost a vlastnosti bude sjednocovat. Viz obrázek 3.1.

Obrázek 3.1 Hlavní třídy



Takto nadefinované základní entity, kde Category může obsahovat Category nebo Detail, určují základní stromovou strukturu uložení dat celého CMS. Takto jdou data uložit do několika stromů, hlavní Category je vždy kořenem a Detaily budou listy. Viz obrázek 3.2.

Obrázek 3.2 Uložení dat



Tyto dva základní typy se pak budou vybírat při tvorbě konkrétní kategorie či detailu. K tomu slouží průvodce pro vytvoření. Mou základní myšlenkou bylo, aby šlo při vytváření přidávat už implementované součásti neboli pluginy např. foto galerie, WYSIWYG, kalendář atd. Tímto pak půjde velice rychle sestavit libovolná položka v systému a práce se urychlí. Pro každou položku vzniká konkrétní záznam v databázi, při vytvoření přes průvodce se vytvoří potřebné tabulky a relace mezi nimi. Jak jde vidět na E-R diagramu obrázek 3.3, každá tabulka má povinné atributy, které se přidávají automaticky a poté přibudou ještě uživatelsky definované. Dále se tabulka vytváří ve všech jazykových verzích a jedné nejazykové, přičemž příslušnost k těmto skupinám lze zvolit u každé přidávané položky tabulky. Vše je v relaci se základním jádrem, aby bylo možno zajistit správu těchto jazykových verzí. Při tomto rozvržení lze také jednoduše vytvářet duplikáty položek. Kde jádro položky, které obsahuje například umístění ve stromě, bude různé, ale všechny budou ukazovat na jednu instanci.

Jakmile máme takto definovanou organizaci a strukturu dat, je možné pomocí modulů zajistit další operace s těmito daty. Nebo jakékoliv modulární rozšíření. Použil jsem takovýto hybrid objektového systému založeného na pluginech a už definovaných komponentách a modulárního systému aby byla možná co největší rozšiřitelnost.

Úvod do implementace

Nastínil jsem tedy základní organizační model mého CMS, v dalších podkapitolách se podíváme na konkrétní implementaci některých funkcí s ohledem na problémy, které se vyskytují při používání AJAX.

Kde napřed probereme hlavní třídy jako základ systému. Jsou to **Category** a **Detail** a jejich předek **Core**. Jejich funkce pro různé výpisy, ukládání nebo vyhledávání, poté i výpis ve kterém lze řadit položky podle atributů, které jsou stejné u všech položek v kategorii. Uvedu i příklad poskládání jednoho ze složitějších SQL dotazů, pro ukázku jak se spojí jádro s detailem, kde detail může být libovolná tabulka.

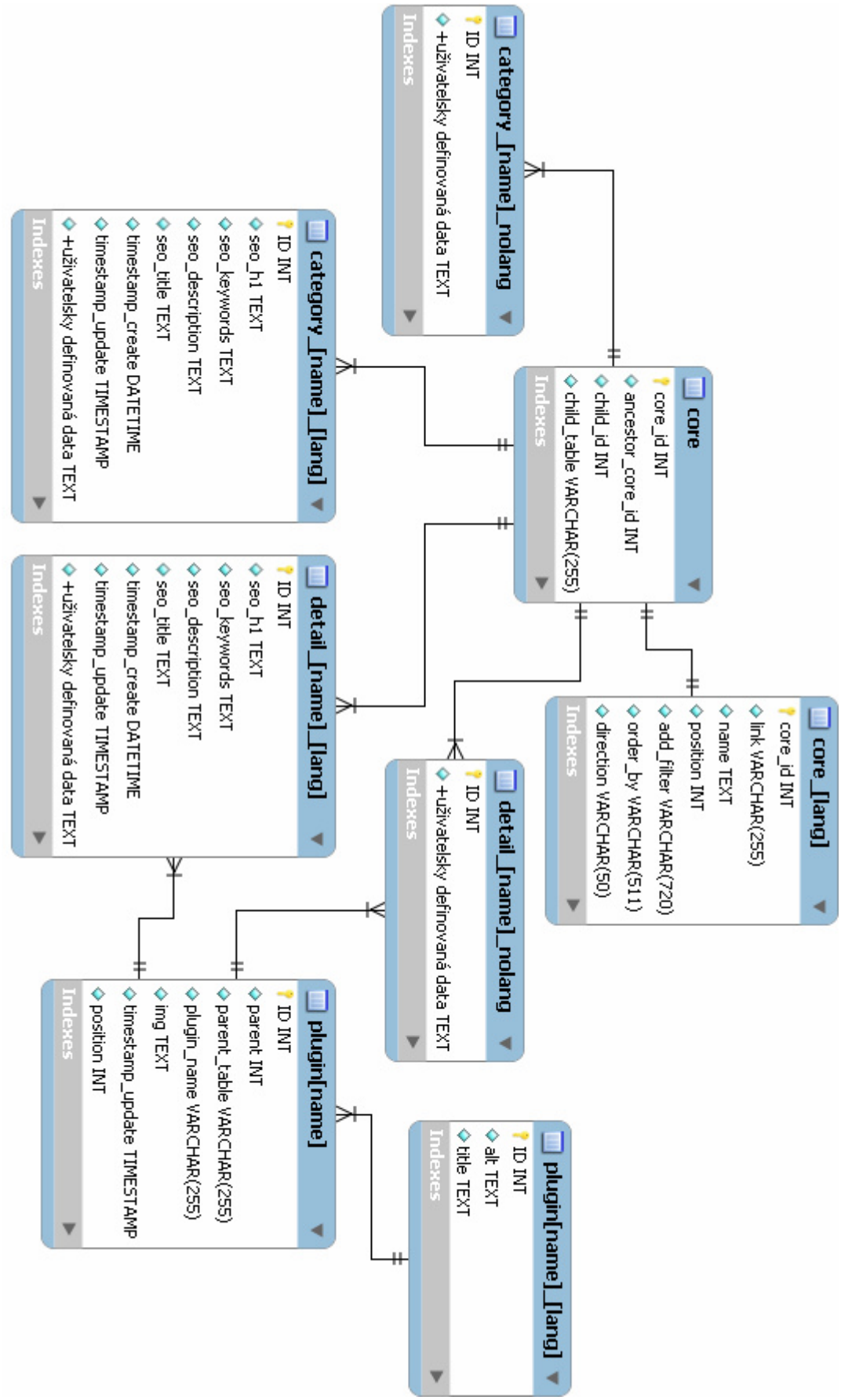
Dále **Creator** pro vytváření položek, který obsahuje vytváření konkrétních Category či Detail, zajišťuje také jejich úpravu a změny v databázi, které s tímto souvisí. Třída **Layout** slouží pro konečné rozvržení těchto položek, pro část prezentační, která je přístupná pro návštěvníky, je to pak tato třída, která zajišťuje sestavení stránky podle URL. Jako nejvhodnější popis funkčnosti se mi zdá popsat přímo **nejdůležitější funkce(metody) a atributy**, jako nadpisy volím přímo názvy funkcí, které zároveň vystihují funkčnost.

Dále rozebrání hlavních pluginů. Zatím je implementovaný pouze plugin pro obrázkové galerie, tento stejný se pak použije pro videa a libovolné soubory.

Uvedu i kousek klientské část přes JScript. Ale spíše jen problémy týkající se uvolňování paměti po drag & drop objektech a jiných. JScriptové funkce musejí při znovu načítání kontrolovat jestli jsou všechny objekty řádně uklizeny.

Kapitola další vývoj obsahuje náčrt nebližšího rozšíření, které se bude řídit požadavky klientů.

Obrázek 3.3: E-R diagram databáze



3.1 Základ systému (třídy)

Třídy se rozdělují pro webovou a administrační část, webová část je v podstatě zjednodušená část administrační. Odpadá zde totiž většina metod pro manipulaci s daty, zbývají tak většinou jen funkce pro výpis.
Jeden velký rozdíl v těchto třídách je ve stylu výpisu, jelikož administrační část je tvořena přes AJAX, vždy se vytvořením objektu nějaké třídy vytvoří jen malá část webu, která se odesílá. Naproti tomu ve webové části se podle URL vytvoří celý web znovu a je poskládaný z několika objektů. Vyplývá zde tedy nutnost mít možnost předávat si hlavní objekt SMARTY, tak aby data, která už v něm jsou načtená, šla použít kdekoliv na webu. Například <title> v hlavičce webu, načte se vždy v objektu detail, ale jeho zobrazení je potřeba až v hlavním layoutu.

3.1.1 Core

Tato základní třída je jádrem pro kategorie a detail, obsahuje jejich stejné metody, například konstruktor, nebo metodu pro ukládání. V této třídě jsou také metody pro obsluhy pluginů, tak aby je mohly užívat všechny třídy zděděné.

3.1.1.1 Atributy

public $dont_look

array("ID","core_id", "ancestor_core_id", "child_id", "child_table", "position", 'order_by', 'direction', 'add_filter');
Používá se pro vyhledávání a pro razeni, určuje podle kterých položek se nemá řadit a ve kterých se nemá vyhledávat

public $smarty

Do tohoto atributu se v konstruktoru vytvoří objekt Smarty. Ve webové části se vytvářet nebude pokud se předá objekt Smarty jako parametr konstruktoru

public $user

Do tohoto atributu se v konstruktoru přiřadí objekt user. Mimo jiné jsou v něm informace o právech uživatele, výchozím jazyku, seznamu všech jazyků atd...

public $core

Zde se v konstruktoru přiřadí ID tabulky core.

public $ancestor_core

Zde se v konstruktoru přiřadí ID předka tabulky ancestor_core_id.

public $child

Zde se v konstruktoru přiřadí ID potomka některé z tabulek s prefixem „detail_“ nebo „category_“.

public $template

Zde se v konstruktoru přiřadí název tabulky potomka, což je zároveň i název šablony, kterou potomek používá pro své zobrazení.

public $tree

Do tohoto pole se vloží cesta ke kořenové kategorii. V další verzi projektu to bude cesta po zarážku, tedy zarážka v kategorii, kterou už nechceme brát v úvahu.

public $db_core_info

Do tohoto pole se vkládá spojení tabulek core a core_[lang], podle právě zvoleného jazyka.

3.1.1.2 Metody

_construct

Formát volání: __construct($ancestor, $core, $template = "")

Konstruktor společný pro třídy category a detail, zadává se nejlépe ID předka a ID core, ovšem stačí pouze core_id, parametr template se použije, pokud chceme zobrazit data do šablony s jiným názvem, než je název tabulky ve které je detail uložen.

save

Formát volání: save()

Funkce pro ukládání nebo vytváření kategorie či detailu. Při vytváření kategorie se dědí vlastnosti s kategorie nadřazené, a to podle čeho se bude v kategorii řadit, směr řazení a filtr přidávání (filtr určuje, které položky lze do kategorie přidat). Také jazykové verze se vždy vytváří takové, ve kterých je nadřazená kategorie.

get_sql

Formát volání: get_sql($table)

Tuto funkci používá funkce pro ukládání. Sestaví část SQL dotazu, ze zadaného názvu zjistí jméno tabulky, pak porovná stejné názvy z databáze a toho co je v $_POST a sestaví řetězec těchto dat pro dotaz. Vše se děje automaticky, v názvu proměnné je uveden její typ, tedy tato funkce rozhodne co s daty udělat než se uloží do řetězce pro SQL dotaz, například aplikovat addslashes(), zpracovat checkbox atd.

get_tree

Formát volání: get_tree($act_core_id)

Zadá se aktuální ID core, vrátí se pole, které obsahuje cestu až k rootu kategorie.

ajax_upload

Funkce pro upload souboru, vygeneruje unikátní název souboru. A uloží ho do databáze podle informací zadaných v $_POST.

handle_plugins

Tato metoda se pro obsluhu pluginů, jako je galerie (ne běžné textové prky). Metoda se zavolá při prvním zobrazení detailu nějaké položky, zjistí jaké obsahuje pluginy a postupně je všechny zobrazí do šablony.

ajax_handle_plugin

Potom co byly pluginy zobrazeny, např. galerie obrázků. Už se zachází s každou galerií zvlášť, tedy vždy se na stránce překresluje jen jeden plugin. Pro zobrazení pouze jednoho pluginu slouží tedy tato funkce.

ajax_plugin_change_position

Pro změnu pozice položky v pluginu, tedy třeba změna pozice obrázku v galerii. Používá se dvou způsobů, buď se obrázky přesunují v maticovém zobrazení, to se rozpozná že v $_POST je vypněno from a to, tedy jen odkud a kam se přesouvá. Jiný mód je pomocí sortable seznamu, tam se vždy odešle celý seznam identifikátoů obrázů a jejich pozic.

ajax_plugin_delete_element

Smazání položky v pluginu.

ajax_save_plugin

Jedná se o uložení informací o pluginu, např. obrázky obsahují jazykové texty title a alt, které lze editovat.

pluginimagegallery_show

Formát volání: pluginimagegallery_show($lang, $plugin_name,$pg = 0)

Funkce pro zobrazení obrázkové galerie, první parametr je aktuální jazyk, druhý název pluginu a třetí je stránkování.

make_aliases

Formát volání: make_aliases($name_array)

Protože v systému je hodně věcí generovaných, např. možnosti přidávání do kategorií, řazení seznamů podle zvolených sloupečků v tabulkách atd., musí existovat tabulka aliasů pro tyto názvy, což je zároveň i filtr pokud nechceme nějakou takovou položku používat, její alias nevyplníme.

ajax_inherit_core_info

Metoda pro manuální šíření atributů, které jsou v tabulce core_[lang]. Rekurzivně volá funkci inherit_core_info($ancestor_core, $type, $inherit_langs, $sql).

check_available_langs

Metoda pro zjištění v jakých jazykových verzích je položka uložena.

change_available_langs

Metoda pro odstraňování nebo vytváření jazykových verzí. Jazykové verze jdou vytvořit pouze takové, ve kterých je dostupná nadřazená kategorie. A pokud se tato funkce použije na kategorii, změny se šíří dolů celým stromem.

3.1.2 Category

Obecná kategorie do které jde vložit jakákoliv položka systému. Obsahuje výpisové metody pro jednu či více úrovní. Výpis celého stromu se zadaným předkem se použije například v mapě webu. Dále obsahuje metody pro vyhledávání, jelikož výsledkem vyhledávání je seznam položek podobně jako kategorie.

3.1.2.1 Metody

show

Metoda pro zobrazeni detailu kategorie.

delete

Smazání kategorie a všeho co je do této kategorie vnořené.

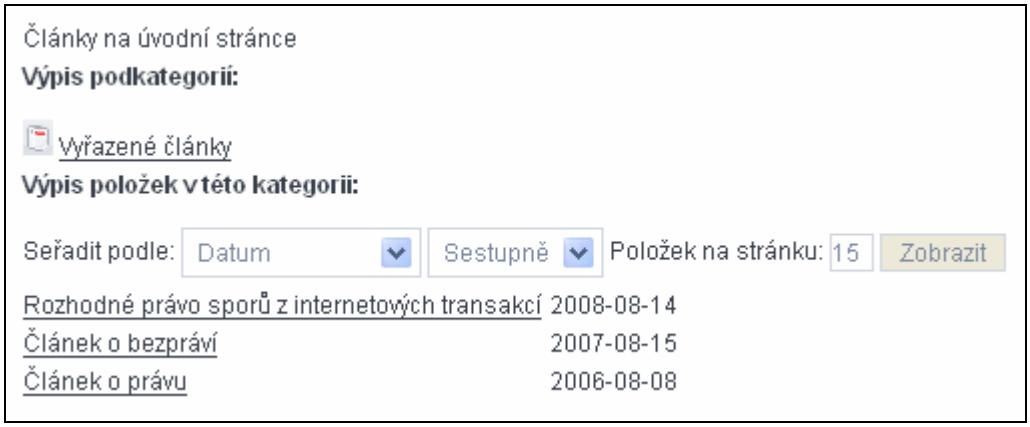
show_list

Funkce pro vypsání seznamu položek v kategorii. Tato funkce vypisuje pouze položky řazené podle pozice, nemá žádné stránkování, používá se výhradně pro výpis v levém menu a spojuje pouze tabulky core a core_[lang] nikoliv vlastní detail položky. Vypisuje vždy bezprostřední potomky dané kategorie.

Má úvaha je taková že stránkování není důležité, protože pokud bude mít kategorie tolik položek, že by vypršel limit PHP při načítání, tak se už nebude řadit podle manuálně určené pozice, ale podle jiné položky, což už se v tomto menu nezobrazí, ale zobrazí se v obsahové části viz. metoda *show_content_list*, zobrazí se pouze vnořené kategorie aby do nich šli vkládat nové položky.

show_content_list

Obrázek 3.4: Výpis kategorie do obsahové hlavní části:



Jak je vidět na obrázku lze si v této funkci zvolit počet položek na stránku, pokud celkový počet tento počet přesáhne zobrazí se stránkování.

Dále si je možné vybrat podle čeho se bude řadit kategorie, přičemž defaultní hodnota je uložena přímo v kategorii. Výběr možností řazení se generuje automaticky, projdou se všechny druhy položek v kategorii a vyberou se sloupečky, které jsou stejné, poté musí být ještě sloupečku zadán alias a je možné si ho vybrat pro řazení.

Tato funkce už tedy spojuje tabulky core, core_lang, detail_[name]_[lang] a detail_[name]_[nolang]. Přičemž tabulek typu detail může být obecné množství.

Uvedu příklad výpisu této funkce:

Jako první zjistíme jaké druhy tabulek typu detail jsou v dané kategorii a vložíme je do pole $detail_tables_in_category. Potom provedeme sestavení dotazu ze zmíněných 4 tabulek, přičemž načítáme pouze sloupečky, které mají všechny tabulky stejné, pozn NATURAL LEFT JOIN se používá proto, že tabulka _nolang se nemusí vyskytovat. Nakonec se musí tyto částečné dotazy spojit pomocí UNION.

```
$count_all = count($detail_tables_in_category);
$count = 0;
foreach ($detail_tables_in_category AS $index=>$detail_table)
{
    $count++;
    $sql_for_detail .= "SELECT $same_collums FROM (SELECT * FROM core
```

```
        NATURAL JOIN
        core{$this->user->content_lang}
        JOIN
        `{$detail_table}{$this->user->content_lang}`
        ON core.child_id = {$detail_table}{$this->user->content_lang}.ID
        NATURAL LEFT JOIN
        `{$detail_table}_nolang`
        WHERE core.ancestor_core_id={$this->core} AND core.child_table
        LIKE 'detail\_%'
        ";
    $sql_for_detail .= ($count == $count_all) ? ") AS {$detail_table}_pom"
: ") AS {$detail_table}_pom UNION ";
}
```

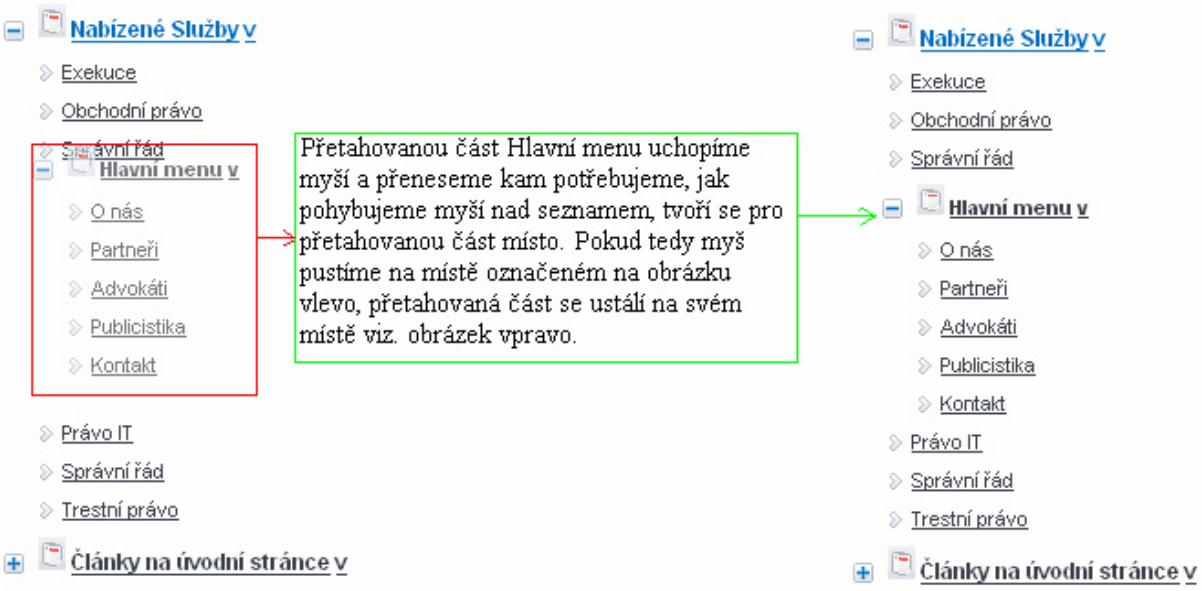
Aby to vše fungovalo, musí se ještě výsledný dotaz takto obalit dalším příkazem SELECT. Poté už jde zadat stránkování pomocí LIMIT a řazení pomocí ORDER BY.

```
$res_detail = q("SELECT * FROM (
    $sql_for_detail
    ) AS main
    ORDER BY {$order_by} {$direction}
    LIMIT {$start}, {$limit}
    ");
```

change_placement

Metoda pro změnu pozice v levém menu. Volá rekurzivní funkci change_placement_rec s parametry $ancestor_core, který udává jaké úrovní stromu se budou přepisovat pozice a $arr což je pole identifikátorů, jehož uspořádání je stejné jako uspořádání stromu. Celé menu je realizované pomocí objektu **sortable**, změna pozic probíhá kompletně na pozadí, na popředí pracuje pouze javascript. Jak ilustruje obrázek 3.5, je možné přetahovat jednu položku nebo klidně celou otevřenou kategorii. Celý strom má omezení pouze pro vertikální pohyb pro lepší manipulaci.

Obrázek 3.5



ajax_find_results

Metoda pro **fulltextové vyhledávání na webu**. Prohledá všechny tabulky a sloupečky, které jsou povolené. Poté ještě označí kontext v hledaném textu.

Zde se vyskytuje hned několik problémů. Například formát hledaného řetězce, pro hledání používám LIKE v sql dotazu, tím pádem se musejí escapovat znaky „%“ a „_“, které jsou součástí jazyka.

```
$query_for_db = addcslashes($query_for_db, '_%');
```

Dále klasické escapování znaků jako jsou uvozovky atd. s porovnáním na magic_quotes_gpc konstantu, jelikož vyhledávaný řetězec se posílá přes $_POST, jež tato konstanta ovlivňuje.

```
...
if (!get_magic_quotes_gpc())
{
    return addslashes($string);
}
else
{
    return $string;
}
...
```

Při vyhledávání v textu uloženém ve WYSIWYG editoru (konkrétně TinyMCE), musím vyhledávaný řetězec zakódovat do HTML entities, protože si tento editor takto uchovává svůj obsah.

```
htmlentities($query,ENT_QUOTES,'UTF-8')
```

Samotný **dotaz pro vyhledávání** se pak poskládá z několika dotazů spojených pomocí UNION, podobně jako v metodě show_content_list, poté lze takový dotaz lehce stránkovat nebo seřadit.

Pro označení kontextu pomocí, které provádím pomocí **regulárního výrazu**, je pak potřeba escapovat znaky regulárního výrazu.

```
$query = quotemeta($query);//escape problemových zn. . \ + * ? [ ^ ] ( $ )
```

Vlastní **označení textu** pak probíhá takto. Z html textu odstraním všechny HTML tagy, pokud by se tagy nechali, musely by se najít vždy i tagy ukončovací, aby se kód nerozházel. Ale jen pro označení kontextu, bude stačit prostý text bez tagů. Poté se provede výřez na zhruba 50 znaků dopředu a dozadu, tak aby se znaky neusekly v půlce. A nakonec se hledaný text zvýrazní.

```
function make_context_tag($query, $tcontent)
{
    $unescape_query = $query;
    $query = quotemeta($query);// escape problemovych znaku . \ + * ? [ ^
] ( $ )

    $tcontent = preg_replace("/\<.+?\>/i", "", $tcontent);

    // vzor pro zamezení useknutí znaku v půlce při použití utf-8
    $pattern = '/(^|\s){1}(.{0,50}'.$query.'.{0,50})($|\s){1}/is';  //
nalevo i napravo musí končit mezerou nebo ^$

    @preg_match($pattern,$tcontent,$matches);

    $vyrez = preg_replace("/$query/i", "<strong>$unescape_query</strong>",
$matches[2]);
    return $vyrez;
}
```

3.1.3 Detail

Objekt třídy detail můžeme zařadit kamkoliv do struktury. Třída obsahuje klasické metody, jako show a delete.

3.1.4 Layout

V administrační části tato třída používá metodu show_page pro zobrazení první strany, poté už se vše načítá pres AJAX.
Ve webové části pak tato metoda přebírá URL a sestaví podle ní celou stránku, kterou následně zobrazí.

3.1.5 Creator

Template creator, nástroj na tvoření a modifikaci šablonek, tedy můžeme přímo ve webovém rozhraní upravovat strukturu detailů a kategorií. Tato struktura se pak promítne do následující úpravy databáze, tedy vytvoří odpovídající tabulky v potřebných jazykových verzích, při úpravě pak kontroluje, které sloupečky tabulek jsou nové a které zmizeli. U každé položky jde zadat zdali má být brána jako nejazyková či jazyková. Následuje postup vytvoření nové šablonky.

Vytvoření obecné šablony

Před přidáním položky do šablony se musí vyplnit například, její název, její název databázi, default hodnota v databázi aj., podle typu položky. Tyto data se pak musí převést do obecné šablony této položky a udělat z ní tak položku konkrétní. Zavedl jsem pouze jednoduchou syntax.
Př. položka WYSIWYG editor:

Takto vypadá obecná **šablona TinyMCE editoru**. Formát proměnných má stejnou syntax jako Smarty. Proměnné se nahrazují jen základní sadou regulárních výrazů. Rozděluji je na proměnné začínající prefixem „el_“, což jsou různé nastavení položky, zde třeba výška a šířka editoru. A poté systémová proměnná „db_col_name“.

Pokud je zadaná jako {$db_col_name} nahradí se za {$jmeno_sloupecku}, tedy za proměnou formátu Smarty, tak aby se do ní mohl poté načíst obsah databáze. Tato stejná proměnná s postfixem _show vypíše jméno sloupce pouze textově. Tohoto se využívá pro identifikaci objektu. Ve složitějších šablonách se pak mohou přidávat ještě další postfixy a celý název bude pak identifikátorem nějakého elementu, např. {$db_col_name_show}_seznam, pak se pomocí AJAX bude přepisovat tento seznam, nebo jiný podle toho, které položce patří.

```
<tr>
    <td>
        {$el_name}
    </td>
    <td>
        <textarea id="{$db_col_name_show}" name="{$db_col_name_show}"
                  style="width: {$el_width}px;  height:{$el_height}px">
                  {$db_col_name}
```

```
        </textarea>
        {literal}
        <script type="text/javascript">
        tinyMCE.execCommand("mceAddControl", true, "{$db_col_name_show}");
        </script>
        {/literal}
    </td>
</tr>
```

Uložení konkrétní šablony

Zdrojový text konkrétních položek je uložen v textarea, tyto texty se poté projedou iterátorem a vloží do souboru. Vzniknou 2 soubory, šablona pro klasické prvky a šablona s pluginy.

Položky jsou zobrazeny jako HTML seznam, který je navíc sortable, aby šla měnit pozice položek. Tento celý HTML seznam se ukládá do souboru a znovu se celý načte pokud budeme chtít šablonu znovu editovat.

Použití šablony

Po uložení šablony se vytvoří nový detail nebo kategorie, kterou lze přidat kamkoliv do struktury. Je automaticky vázána na svou zobrazovací šablonu.

3.1.6 User

Objekt uživatel se vytváří vždy, v konstruktoru je přímo kontrola přihlášení. Dále obsahuje informace o tom jaký jazyk uživatel právě používá a seznam dostupných jazykových verzí. Dále také práva, která se rozdělují zatím pouze a na uživatele a superuživatele.

3.2 Pluginy

Za plugin v CMS považuji až větší samostatné části, které mají nějakou zapouzdřenou funkcionalitu, tedy práce na stránce probíhá pouze s jedním tímto pluginem a to pomocí AJAX, obnovuje se tedy jeho část nebo maximálně celý plugin. Zatím jediný takovýto realizovaný plugin je Obrázková galerie, která je ovšem při malé úpravě už i galerií souborů nebo videí, jediný rozdíl mezi nimi je jiná forma prezentace, práce s daty zůstává stejná.

3.2.1 Plugin image gallery

Obrázková galerie se rozlišuje na dva módy, editační a maticový. Upload souboru je realizovaný pomocí IFRAME, na onload událost IFRAME se pak reaguje funkcí, která překreslí galerii.

3.2.1.1 Maticový mód

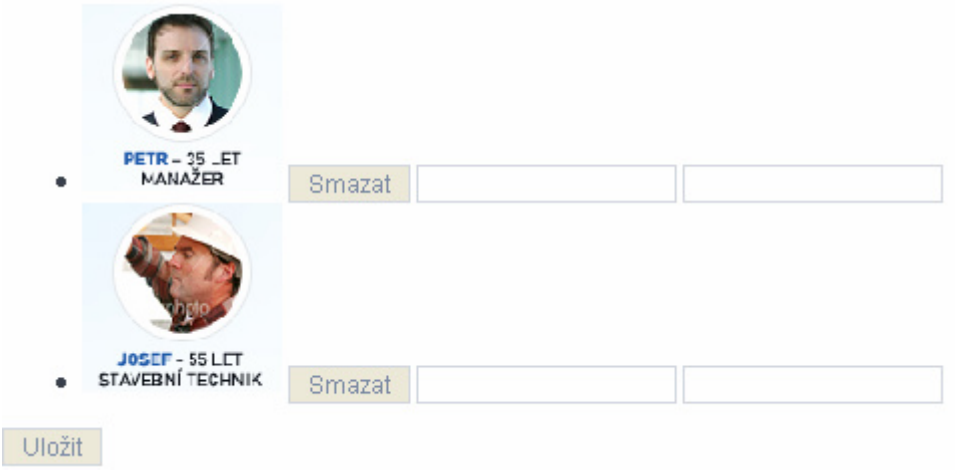
Slouží zejména pro rychlou změnu pozic obrázků. Uspořádání v matici je pro změnu pozic vhodné. Jak je zřejmé na obrázku 3.6, při přetáhnutí obrázku na pozici kde ho chceme vložit, se tato pozice zvýrazní a místo pro vložení se roztáhne. Po vložení obrázku proběhne změna pozic a celá matice se načte znovu přes AJAX. Plánované rozšíření je přesouvání pouze ve frontendu a změna pozic v databázi pouze na pozadí. Ovšem i tato varianta je rychlá, při dalším zobrazení se už obrázky stahují z cache prohlížeče.

Obrázek 3.6: Maticový mód.

3.2.1.2 Editační mód

Pro změnu pozic používá sortable, tedy změna probíhá kompletně na pozadí. Obsahuje navíc funkci pro smazání obrázku. Poté se zde editují jazykové texty alt a title jednotlivých obrázků.

Obrázek 3.7: Editační mód.



3.3 Rozhraní klienta přes Javascript

Tedy programování na klientovi. Popíšu zde nejdůležitější funkce pro volání PHP skriptů přes AJAX. jelikož většina freeware doplňků jako WYSIWYG editor, či drag & drop objekty počítá s vždy s obnovou celé stránky a tedy s uvolněním paměti, při práci přes AJAX musí programátor toto zvládnout sám. Stručně popíšu sadu funkcí pro načítání stránek, ukládání formulářů, obsluha pluginů a jiné, přičemž důraz bude kladen právě na uvolňování paměti.

3.3.1 Funkce pro obecné stránky

3.3.1.1 Jazykové verze

Obsahuje funkce pro změnu jazykové verze, jak hlavního stromového menu, tak hlavní části CMS systému. Tato funkce si bere URL, kterou má znovu načíst ze zdrojového kódu v novém jazyce, kde se ukládá funkcí pro zobrazení. Ukládají se ULR panelů „Výpis kategorie“ a „Detail“.

3.3.1.2 Hlavní stromové menu

Funkce pro obnovení části jedné úrovně hlavního stromového menu, tato funkce se volá pokaždé když dojde k uložení nějaké položky, **obnoví se vždy kategorie, ve které se položka nachází**. Zavolá se také při otevření nějaké kategorie, uzavření kategorie už pak probíhá pouze odmazáním kousku seznamu v DOM. Vždy se musí znovu vygenerovat celý Sortable seznam těmito dvěma příkazy:

```
Sortable.destroy('levemenu_id');
create_tree("levemenu_id");
```

Funkce create_tree v sobě obsahuje vytvoření sortable a vazbu na AJAX funkci pro změnu pozice položek v tomto menu. Metoda Sortable.serialize pak vytvoří vícerozměrné pole kopírující strukturu stromu a obsahující ID jednotlivých elementů.

```
function create_tree(tree_id)
{
    Sortable.create(tree_id ,
    {
        tree:true,
        scroll:window,
        onUpdate: function()
        {
            new Ajax.Request("index.php?id=category&fce=change_placement",
            {
                method: "post",
                parameters: Sortable.serialize(tree_id)
```

```
            });
        }
    });
}
```

3.3.1.3 Obsahová část

Funkce pro zobrazení v obsahové části, prvním parametrem je URL, která se ukládá i přímo do DOM aby byla dostupná pro načtení této stránky v jiné jazykové verzi, druhým parametrem je pak id elementu, do kterého se bude obsah vkládat. Tento parametr se použije i pro zaktivnění panelu Výpis kategorie nebo Detail, podle toho jaký typ výpisu byl žádán. Tato funkce obsahuje funkci pro odstranění všech instancí TinyMCE WYSIWYG editoru, včetně elementů s třídou contextMenu, které si TinyMCE vytváří.

```
function mceRemoveAllInstances()
{
    for (n in tinyMCE.instances)
    {
        inst = tinyMCE.instances[n]; // výběr všech instancí

        if (!tinyMCE.isInstance(inst))
            continue;
        tinyMCE.removeInstance(inst);// pokud je instance, odstraní
    }
    var tinyMCEcontextMenu;
    while(tinyMCEcontextMenu = $('main_body').down("div.contextMenu"))
    { // vybere všechny contextMenu a odstraní
        tinyMCEcontextMenu.remove();
    }
    tinyMCE.idCounter=0; // nakonec reset počitadla instanci TinyMCE
}
```

Dále je důležité z obsahové části **odstranit droppables** před změnou, protože elementy musí být v době odstraňování ještě přítomny v DOM, aby je metoda remove() mohla odstranit, odstraňují se funkcí scriptaculous_remove_droppables('', "dropsite"). Tato funkce odstraní všechny droppables, které se v DOM vyskytují a mají třídu dropsite. Tato funkce se používá i u pluginů, ale má zadán první parametr, který omezí mazání jen na potomky zadaného uzlu DOM.

```
function scriptaculous_remove_droppables(parent_el, el_class)
{
    if (parent_el)
    {
        var divs = $(parent_el).select(el_class);
    }
```

```
    else
    {
        var divs = $$(el_class);
    }
    for (var j = 0; j < divs.length; j++)
    {
        Droppables.remove(divs[j]);
    }
}
```

Mazání draggables by mělo proběhnout při načítání nové stránky též. Při přechodu na novou stránku chci smazat všechny draggables kromě těch co jsou v hlavním stromovém menu. Algoritmus co používají je podobný, jen mažu draggables, které patří danému pluginu, jiné ne.

```
Draggables.drags.each(function(d)
{
    var e = $(d.element);
    if (d.element && e && e.up(".levemenu_class"))
    { // pokud je draggable ve stromovém menu, neudělám nic
        ;
    }
    else
    { // jinak draggable smažu
        e = d.element;
        d.destroy();
        if (e.parentNode) {e.parentNode.removeChild(e)};
    }
})
```

Poté je zde několik funkcí pro ukládání stránek, liší se odesíláním formulářů. Buď stačí klasická serializace formulářových prvků, nebo složitější přepisování HTTP metody odesílaní kvůli TinyMCE editoru. Další odlišnost těchto funkcí je, jestli vrací svůj výsledek do elementu, alertem a nebo a nebo se zavolá další funkce, která byla určena pomocí PHP skriptu.

3.3.2 Funkce pro pluginy

Funkce pro uvolňování paměti drag & drop už byly popsány. Zde se vztahují vždy jen k jednomu pluginu, a to pomocí id jeho hlavního elementu.

Dále obsahuje funkce pro mazání jednotlivých položek v plugin, posunování položek v pluginu, inicializaci drag & drop objektů, případně celého Sortable stromu. A nakonec funkci pro zobrazení jednotlivého pluginu.

3.4 Další vývoj

Zde bych rád probral nejbližší vývoj aplikace, tedy věci se kterými jsem počítal a aplikace je na ně připravena.

Upravení obrázkové galerie, na galerii obecných souborů, galerii videí a to jak embed objektů, tak například youtube formátu atd.

Duplikace položek, správa duplikací a umístění duplikátu kamkoliv do stromu. Duplikace proběhne pouze zkopírování core části, odkaz na detail bude pořád stejný.

Relační plugin, tento plugin bude obecným pluginem pro vytváření vztahů detailu s dalšími položkami ve stromu. Při vytvoření pluginu přes průvodce se zadá jeho hlavní kategorie a další parametry, bude například ještě obsahovat 2 textové parametry a jeden checkbox, zde půjde přidávat neomezeně z formulářových prvků. Po vytvoření položky, která obsahuje takovýto plugin, pak bude možné do této položky přidat libovolný počet relací a zadat jejich parametry. Například takto může fungovat nějaká sada produktů, relačním pluginem se do sady přidají jednotlivé produkty, přičemž každý může mít další parametry, jako kolik kusů od jednotlivého produktu se v sadě nachází atd.

A jako vzdálenější vývoj pak bude oddělení eshopu a prezentace. U větších systémů, které mají několik tisíc produktů a několik tisíc článků bude rychlejší mít v databázi tyto dva typy oddělené.

4 Závěr

S programem v této fázi je možno poměrně rychle vytvořit prezentaci. Je připraven na různé změny struktury a obsahu uložených dat, tedy už vytvořená prezentace jde použít znovu jen se změnou CSS. Ale zároveň je připraven na modifikace, které často požaduje klient a vyžadovaly by větší zásahy do kódu. Jelikož klient požaduje změny vícekrát během celkové komunikace, tyto úspory času, vlivem možnosti měnit strukturu přímo v prohlížeči, jsou poté vcelku znát.

Při tvorbě jsem se důkladně seznámil s nejpoužívanějšími programovacími jazyky pro tvorbu webových aplikací, naučil jsem se rychle zakomponovat nejrůznější freeware doplňky, tedy rychleji pročítat manuály. Tyto doplňky jsou pravidelně aktualizovány a lze tak program lépe udržovat. Na druhou stranu je někdy potřeba řešit vzájemnou nekompatibilitu doplňků nebo jejich chyby. Většinou se však dají najít takové, které jsou spolu přímo testovány.

Z hlediska dalšího vývoje lze rozšiřovat hlavní objektový model celého programu o další pluginy s různorodou činností, nebo přidávat klasické moduly a do jádra tak vůbec nezasahovat. Základ systému, tedy jádro, které spravuje a vytváří dokumenty, lze vždy využít znovu, pouze se vybere, které pluginy chceme používat. Samostatné specializované moduly jsou pak většinou dělány na míru danému systému a znovupoužitelnost je pak možná jen ve velmi podobných systémech. Podrobný další vývoj je popsán v kapitole 3.4 a bude se řídit projekty, na kterých bude toto CMS vyvíjeno.

Návaznost mého projektu na ostatní je zřejmá například při užití drag & drop objektů a různých grafických efektů, většina nových komerčních CMS tíhne k tomuto trendu, snaží se být co nejvíce uživatelsky přívětivá a pohodlná při práci.

Literatura

[1] Gilmore, W. Jason: Velká kniha PHP 5 a MySQL, kompendium znalostí pro začátečníky i profesionály. 1, Brno, ZONER software s.r.o. 2005.

[2] Wyke-Smith, Charles: CSS, Využijte kaskádové styly naplno. 1, Brno, Computer Press a.s., 2006.

[3] http://www.wikipedia.org (prosinec 2008).

[4] http://www.prototypejs.org prosinec 2008).

[5] http://script.aculo.us/ (prosinec 2008).

Seznam příloh

Příloha 1

Fungující prezentace na http://www.mitrovic.cz/

Fungující aplikace na http://www.mitrovic.cz/admin
Přístup pro tvůrce webu jméno „testa“ heslo „testa“
Přístup pro uživatele jméno „test“ heslo „test“

Aplikace ještě není v ostrém provozu, je tedy možné si ji vyzkoušet a měnit obsah. Pokud chcete otestovat celkovou funkčnost, přihlaste se jako tvůrce webu.

Instalace
Soubor cms-database.sql nahrát do mysql databáze.
Obsah adresáře „cms“ nahrát na apache server. V tomto adresáři nastavit správnou cestu pro .htaccess a přístupy do databáze v souboru _settings.inc . Poté ještě nastavit práva pro adresář uploadedfiles a adresář tinymce pro upload souborů.