Tvorba internetových aplikací pomocí Rich Internet Application AJAX

Bakalářská práce

Ondřej Ašenbryl

Vedoucí bakalářské práce: PaedDr. Petr Pexa

Jihočeská univerzita v Českých Budějovicích

Pedagogická fakulta

Katedra informatiky

2008

Prohlášení

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách.

V Českých Budějovicích dne

Anotace

AJAX je moderní a hodně se rozšiřující technologie. Je relativně nová, proto se v této práci budu snažit shrnout její nejdůležitější rysy, předvést její funkci, poukázat na možná úskalí a popsat podrobně její implementaci. S AJAXem je spojeno hned několik moderních technologií (pojmů), které jsou na webu využívány. Programátor v AJAXu je musí umět zvládnout a proto se nevyhnu ani poukázání na témata jako JavaScript, DOM, XML a serverové technologie.

Abstract

AJAX is a modern and very expanding technology.It is relatively new, so i will do my best to sum its main features up, show its function, refer to its probable difficulties and describe its implementation in detail. There is a number of modern technologies (conceptions) consolidated with AJAX, which are used on web. Any AJAX programmer must be able to work with them, and so I can´t avoid pointing out subjects as JavaScript, DOM, XML and server technologies.

Poděkování

Rád bych poděkoval vedoucímu mé závěrečné práce PaedDr. Petru Pexovi za to, že mi umožnil se touto tématikou podrobně zabývat, za konzultace a rady, které souvisely s vytvořením náplně práce.

Dále bych chtěl poděkovat rodině za podporu v čase, který jsem této práci věnoval.

Obsah

1. Úvod

Ještě před zveřejněním témat bakalářských prací jsem se s pojmem AJAX několikrát setkal. Nové možnosti, které AJAX nabízí, jsou schopny programátora nadchnout, což byl i můj případ.

První uvědomění si, že něco jako AJAX existuje, jsem získal zamyšlením nad funkcí poštovního klienta mé emailové schránky, který je na této technice postaven. Do té doby jsem netušil jak nějaké věci fungují a když jsem přemýšlel o tom, jak jsou tvořeny, nedokázal jsem přijít na žádný rozumný způsob jejich řešení. Poté co jsem se dopátral, v čem je ten vtip, byl jsem o to víc překvapen, když jsem zjistil, že vedle na první pohled AJAXových funkcí provází pojem AJAX uživatele dnešního internetu doslova na každém kroku. A to i v sebenepatrnějších situacích, což si běžný uživatel ani neuvědomí.

Protože jsem se v té době o vývoj webových aplikací již zajímal, nějaké zkušenosti s tvorbou webu klasickým způsobem pomocí PHP, HTML, CSS a JavaScriptu jsem už nasbíral, zvládl jsem nějaké jednoduché funkční aplikace v AJAXu vytvořit. Výhodou použití některých řešení jsem byl velice mile překvapen. Proto jsem uvítal možnost zabývat se touto tématikou podrobněji v této práci.

AJAX je pojem relativně nový, ale velmi rychle se rozšiřující. Jeho okamžitému porozumění však trochu brání skutečnost, že programátor musí nejprve na nějaké úrovni zvládnout jiné standardně se používající techniky a technologie. To je asi jeden z důvodů, proč AJAX v dnešních dnech není na většině škol zahrnut do běžné výuky.

Proto bych chtěl tímto textem přispět k jeho představení, rozšíření, objasnění a k jeho stále vzrůstající oblibě. Čtenář bude veden od začátku a text bude postupně doplňovat důležité informace, které umožní s tímto pojmem spolupracovat.

Jsem si jistý, že neznalý čtenář bude překvapen, jakých možností se mu za

minimální úsilí nabízí a využije nově nabytých schopností.

Avšak i přesto, že zmíním i pojmy, na nichž je AJAX založen, nechávám na jiných zdrojích, aby podrobně vysvětlily, jak se pracuje s JavaScriptem, PHP, CSS, MySQL a HTML. Jak už jsem zmínil, chce to přeci jen nějakou praxi a komplexní popsání by bylo nad rámec této práce.

Jako praktickou aplikaci k textu jsem si vymyslel program, který je na AJAXu celý postaven. Jedná se o webové rozhraní, které umožní univerzální správu kategorií článků a fotogalerií. Jak plánuji do budoucna, stane se hlavní součástí administračního rozhraní internetové prezentace Českých Budějovic, kterou chci pojmout jako nekomerční projekt, se snahou vytvořit fotogalerii různých částí města. Tento záměr mám již delší dobu, hlavně proto, že se na internetu žádné takové materiály nenachází. Proto je celý návrh koncipován na míru mého úmyslu.

1.2. Náhled do historie webových aplikací

Internet byl nejprve určen pouze pro vědecké účely amerických univerzit. Za dnešní masivní rozmach může značkovací jazyk HTML (Hypertext Marup Language), který tvoří kód webové stránky a je interpretován webovým prohlížečem.

HTTP (Hypertext Transfer Protocol) je do dnešních dnů využíván jako hlavní protokol pro přenos dat Internetem. Jeho vznik je datován do roku 1991. Tento protokol nejprve jen otevíral a uzavíral spojení. Dnešní verze 1.0 je vytvořena v roce 1996 a má podporu ve všech moderních webových prohlížečích.

První webové aplikace byly statické, tedy fungovaly zhruba jako noviny, ve kterých si čtenář listuje. Uživatel odeslal HTTP požadavek a přišla mu stránka ze serveru. Do obsahu nemohl zasahovat. Postupem času (jak začínalo být nutné) se stránky potřebovaly dynamizovat a umožnit tak uživateli určité

zásahy do běhu stránky.

V polovině devadesátých let se jako první řešení začalo používat CGI (Common Gateaway Interface). Od klasického modelu bylo možné na severu po odeslání HTTP požadavku spouštět programy, které stránky nejprve vyvořily a poté je odeslaly klientovi. Bylo možné načítat a ukládat data do databází, nebo reagovat na akce uživatele, který mohl zasáhnout přímo do obsahu stránky. CGI však má svá úskalí, hlavně co se týče zabezpečení.

Další možností dynamizace jsou Applety z roku 1995. Applety vyžadují použití prohlížeče umožňujícího pracovat s jazykem Java od společnosti Sun, jehož interpret má uživatel nainstalován. Prvním prohlížečem, který s Applety pracoval, byl Netscape Navigator. Applet je většinou menší program, který je implementován přímo do stránky. Problém Appletů je pro uživatele bohužel ten, že pro správnou funkčnost je potřeba správné verze jazyka Java v počítači. Toto je asi důvod, proč Applety nedosáhly tak masivního rozmachu.

Další možností dynamizace je JavaScript od společnosti Netscape. Je vytvořen pro dynamizaci stránky na straně klienta. Tedy proto, aby uživatel svým chováním mohl v už jednou načtené stránce měnit tagy a se zrodem DOM (Document Object Model) může JavaScript prakticky pracovat s kompletní stránkou v prohlížeči, reprezentovanou objektovým modelem. I když byl JavaScript často zavrhován a ze začátku neměl moc dobrou podporu v prohlížečích, dnes je toto téma žádané a jak ukazuje tato práce, JavaScriptem je možné dělat velké a moderní věci.

Dnes jsou nejrozšířenější weby pracující s technologiemi na straně serveru. Je jich celá řada: ASP, PHP, Servlety, JSP…Pomocí nich lze generovat dynamicky HTML kód, který je poslán uživateli. Je zde možnost provádět složité akce a využívat mnoho možností, které nové programovací jazyky nabízejí.

Jako další pokus o vytvoření kvalitního prostředku pro dynamizaci přišla firma Macromedia se svým produktem Flash. Flash nabízí velmi dobré možnosti grafického vyjádření a aplikace v něm vytvořené se v podstatě

podobají běžným desktopovým aplikacím. Flash vyžaduje pro svou funkčnost u klienta (jako Applety) určitý software, s čímž mají uživatelé často problém.

1.3. Rich Internet Application (RIA)

RIA (Rich Internet Application) je nový směr, kterým se začínají ubírat vývojáři webových stránek. Dosavadní postupy, tak jak je každý z dnešního internetu zná, jsou stanoveny technologiemi, které nenabízejí všechny možnosti běžných desktopových aplikací. Proto nové cesty odstraňující různá omezení ve vývoji webů jsou poměrně vítány.

Hlavními důvody pro tvorbu Rich Internet Application jsou zejména:

- Nároky na dokonalejší grafické rozhraní
- Uživatelsky přívětivé chování
- Komfort funkcí odpovídající klasickému desktopovému řešení

Za Rich Internet Application tedy můžeme považovat weby, splňující tyto nároky.

Dnešní webové aplikace musí běžet na protokolu HTTP a pracovat tak na modelu žádost/odpověď, což představuje pro klasicky tvořené weby přeci jen určitá omezení. Dále si programátoři musí vystačit s HTML a CSS a vytvořit grafické vyjádření stránek pomocí těchto technologií. To jsou také určitá omezení [3].

Proto se stále vymýšlejí nové cesty a možnosti, jak vyrobit dokonalejší webové prezentace, splňující pojem RIA.

Vedle AJAXu, jenž je jednou z možností, jak se RIA nárokům přiblížit, je to i například Macromedia Flash, který umožňuje opravdu výborné možnosti grafického vyjádření a nabízí možnost vytvoření velice propracovaných aplikací, které se blíží běžným desktopovým.

2. AJAX

2.1. Co je AJAX?

AJAX je relativně nová technika, která začíná být využívána programátory webových stránek po celém světě. Záměrně zde používám výraz "technika", protože jak již název vypovídá, nejedná se o samostatnou technologii, ale spíše o dovednost postavenou na několika dnes běžných technologiích. Výraz AJAX je akronymem slov Asynchronous JavaScript and XML, tedy česky Asynchronní JavaScript a XML. Je založen primárně na použití JavaScriptu. Pro přenos dat mezi serverem a webovým browserem se pak využívá např. XML, JSON nebo třeba jen holý text.

AJAX spadá do skupiny RIA a dovede webové stránky obohatit o nové možnosti, kterých by programátor do té doby klasickými metodami nedocílil, nebo v nějakých případech jen velice těžko.

Hlavním důvodem, proč se AJAX dnes tak využívá je ten, že programátorovi nabízí možnost zasílat HTTP požadavky a to bez nutnosti načíst znovu stránku.

AJAX je oblíben také i proto, že běžný uživatel je čím dál tím náročnější a na klasické webové stránky klade velké nároky. Vyžaduje stránky podobající se více běžným desktopovým aplikacím. Není ale zvykem vytvářet pro tento účel nějaké spustitelné soubory a kompletní web se dnes kvůli svým nevýhodám ve Flashi již moc nevytváří. Proto možnost použít k realizaci takto náročných systémů AJAX je programátory vítána a AJAXu se tak předpovídá velká budoucnost a stává se z něj doslova fenomén.

2.2. Výhody a nevýhody

Jako každý pojem má i AJAX své výhody a nevýhody. Pokusím se je na tomto místě vyjmenovat a tím i nastínit případy vhodného použití.

2.2.1. Výhody

- *AJAX nabízí jakýsi uživatelský comfort*, urychluje uživateli práci**.** Nemusí se pokaždé znovu nahrávat nová stránka. Tuto vlastnost ocení hlavně návštěvník webu, pro kterého je možné vytvořit přátelské a elegantní prostředí.
- *AJAX šetří datové přenosy*. U klasické webové aplikace se s každým požadavkem musí uživateli posílat ze serveru celý kód stránky, v němž nemusí být oproti poslednímu vyobrazení stránky velký rozdíl. Je tedy zbytečně posíláno velké množství dat. S AJAXem se posílá jen to, o co uživatel svou akcí žádá.
- *AJAX využívá již zaběhlé*, dobře popsané technologie. Proto není programátor nucen učit se od začátku nový jazyk. Toto ovlivňuje rychlost, za jakou je programátor schopen vytvářet funkční aplikaci.
- *Použitím klasických prostředků pro webové aplikace jsou programová řešení relativně jednoduchá* a náklady na softwarové vybavení uživatelů jsou minimální. Uživatel si vystačí se svým operačním systémem a webovým prohlížečem.

2.2.2. Nevýhody

- *Uživatelé jsou zvyklí při navigaci ve webovém prohlížeči používat tlačítko Zpět*. Bohužel tato možnost nám při implementaci AJAXu do stránky odpadá. Jak již bylo řečeno, nepřehraje se celá stránka, ale pomocí JavaScriptu jen její část. Proto se nemění URL stránky a tím při

kliknutí na tlačítko Zpět dochází k návratům, které by uživatel jen těžko mohl očekávat. Tato vlastnost bývá často označována za největší nevýhodu.

- S tímto je úzce spjata další nevýhoda, a to že stránka, na které se uživatel právě nachází v nějakém stavu, se *nedá bookmarkovat* nebo např. někomu poslat emailem.
- Dále není zaručeno to, že fulltextový vyhledávač (kvůli JavaScriptu) musí správně stránku indexovat a nemusí se dostat na všechny její části. Což je v dnešní době, která klade velký důraz na SEO validní stránky, docela problém.
- *Klient může mít JavaScript vypnutý*. Tento krok bude mít za následek, že AJAXová aplikace nebude vůbec fungovat. Proto je vhodné mít v záloze pro tento případ nějaké jiné řešení, aby návštěvník stránky alespoň něco uviděl.
- Další problém může nastat v případě, že náš kód se bude chovat v různých prohlížečích nekonzistentně. Je možné, že pro různé prohlížeče budeme muset psát několik verzí Javasriptu.

AJAX rozhodně není řešením všech problémů na webu a programátor musí počítat s tím, že všechno AJAXem prostě udělat nejde. Někomu by se na první pohled mohlo zdát, že nevýhody nad výhodami drtivě vítězí, ale toto neznamená, že by AJAX nestál za to se ho naučit. Už jsem zmiňoval, že pomocí něho lze vytvořit prostředí uživatelsky velmi přívětivé s prvky, které by na webu jen tak někdo nečekal. I na internetu platí to známé pravidlo „náš zákazník, náš pán“ a to je právě ono. Vždyť pěkné webové prezentace se vytvářejí právě pro lidi a jsou tak vizitkou společností, které dbají na to, aby jejich web splňoval dnešní nároky a zůstával vždy aktuálním.

2.2.3. Příklad vhodného použití AJAXu

Jako nejznámější použití AJAXu je asi tzv. Suggest (Google Suggest), neboli našeptávač. Pomáhá při vyhledávání výrazů v databázi, kdy po napsání části hledaného výrazu bývají automaticky zobrazovány možnosti, které uživatel hledá. Je to vlastně takový filtr dat v databázi.

AJAX bývá nasazován při kontrole dat na serveru, hlasování do různých anket, nebo např. při složitějších aplikacích jako je přetahování položek myší, tvorba grafů a jiných výstupů v reálném čase atd.

Naopak nevhodné je využívat AJAX místo klasických odkazů v menu stránek a spoléhat se pouze na něj. Toto řešení sice přináší výhodu rychlejšího webu, ale problémy s přístupností SEO jsou dnes poněkud závažnější problém.

2.3. Podřebné znalosti pro práci s AJAXem

AJAX není rozhodně určen pro programátory, kteří nemají žádné zkušenosti s tvorbou webu běžným zůsobem. Tedy nemají znalosti HTML, CSS, nějaké serverové technologie a alespoň z části JavaScriptu. To dokládají i různé publikace, které jsou AJAXu věnovány. Zde je popisu těchto technologií věnováno minimum a autoři doporučují nejprve tuto problematiku nastudovat samostatně.

Představíme si zde alespoň stručně to nejdůležitější, co bych také doporučil si prostudovat samostatně, a poté se k AJAXu vrátit.

2.3.1. (X)HTML

HTML je tzv. značkovací jazyk, jež byl vytvořen v roce 1991 jako jazyk určený pro publikování hypertextu na webu. HTML je založené na používání

speciálních značek (tagů). Tagy jsou složeny z názvu, který je uzavřen do špičatých závorek (<,>). Vše ostatní, co není uzavřeno v těchto závorkách, je textová náplň stránek. Tagy určují jaký je význam textu a vytvářejí různé prvky na stránkách. Např. tabulky, odkazy, nadpisy, odstavce, seznamy atd. Tagy jsou buď párové nebo jednoduché. To znamená, že je tag složen ze dvou značek, kde jedna určuje začátek platnosti tagu a druhá platnost ukončuje (např. <h1></h1>).

Tag ukončovací se liší od počátečního tím, že před názvem tagu obsahuje lomítko „/“. Uvnitř párových tagů mohou být i jiné tagy, ne jen obyčejný text.

Tagy mohou mít své atributy. Ty vyjadřují nějakou vlastnost tagu. Atributy se píší do počátečního tagu a každý atribut má svou hodnutu. Hodnota atributu je uzavřena v uvozovkách. Těchto atributů je možno do tagu napsat více, přičemž je oddělit mezerou.

Příklad:

```
<h1>nadpis</h1>
<br>
<div align="left">
      Text zobrazovaný na stránce.
</div>
```

Vývoj HTML sice skončil, ale HTML má své následníky. XHTML je nástupce HTML založený na XML. Odtud plynou jeho přísnější nároky na správně napsaný kód.

XHTML používám ve své praktické aplikaci, proto trochu rozvedu další pravidla, která souvisí s psaním jeho kódu.

Správně napsaný a W3C validní XHTML by měl splňovat tyto pravidla: [7]

- Na začátku zdrojového kódu by měla být klasická XML deklarace s používaným kódováním

```
<?xml version="1.0" encoding="utf-8"?>
```

- Následuje deklarace DTD. Jedna ze tří variant jazyka XHTML 1.0, tedy Strict, Transitional, Frameset. V mém případě varianta Strict.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

- Kořenový element html obsahuje určení jmenného prostoru a jazyku pomocí atributu xmlns.

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="cs"
lang="cs">
```

- Element html vždy obsahuje dva elementy, head a body. Hlavička musí obsahovat element title.
- Všechny tagy i atributy musí být malými písmeny, XHTML je case sensitive.
- Všechny hodnoty atributů musí být v XHTML v uvozovkách.
- Všechny XHTML tagy musí být párové. Při použití prázdného tagu se musí tag ukončit lomítkem, např.
 pro odřádkování
- Tagy se nesmí nikdy křížit.
- XHTML 1.0 Strict neobsahuje žádné atributy sloužící k formátování.
- Vkládané skripty na straně klienta (např. JavaScript) by měly být vloženy do sekce CDATA. Starší prohlížeče ale CDATA nepodporují.

```
<script><![CDATA[kód skriptu ]]></script>
```

- Znak & musí být převeden na html entitu i pokud je součástí URL.

2.3.2. JavaScript

Ovládnutí JavaScriptu je pro programátora AJAXových aplikací klíčovým momentem. Právě z JavaScriptu plyne výhoda aktualizace pouze některé části stránky, protože JavaScript nabízí programátorovi možnosti jak dynamizovat web, který je již načten a interpretován webovým browserem. K jeho správné funkčnosti postačí pouze webový prohlížeč, který má Javasript implementován. Vždy tomu tak nebylo, ale v moderních webových prohlížečích má dnes slušnou podporu a doby, kdy byl zatracován a špatně standardizován již pominuly.

JavaScript navíc pracuje poměrně rychle a s pomocí CSS je mocným nástrojem při kvalitním grafickém stvárnění. Navíc se řadí do kategorie tzv. jazyků typu "C Like" (podobných jazyku C), takže syntaxe JavaScriptu umožňuje používat známé konstrukce, které se využívají ve většině běžných programovacích jazycích.

JavaScript je možné psát v jakémkoli textovém editoru a do (X)HTML kódu je ho možné vložit několika způsoby:

- Kdekoli ve stránce vložením do elementu script

```
<script type="text/javascript">
      zde bude zdrojový kód v JavaScriptu
</script>
```

- Do elementu script pomocí atributu src jako externí soubor. Většinou s příponou js

```
<script type="text/javascript" src="soubor.js"></script>
```

- Pomocí tzv. události JavaScriptu do nějakého tagu ve stránce. Událost se vždy aktivuje, když nastane nějaká akce, na kterou událost čeká.

Např. kliknutí myši, načtení stránky, scrollování, najetí na obrázek. Je jich velké množství (viz. přílohy této práce)

```
<div onclick="zde bude zdrojový kód v JavaScriptu">
      nějaký text v tagu div
</div>
```

Podrobnější představu o zdrojových kódech v JavaScriptu si uděláme přímo na ukázkách AJAXových funkcí dále v textu. Ještě jednou je však nutno říci, že AJAX není moc vhodný pro někoho, kdo nemá alespoň nějaké znalosti JavaScriptové syntaxe.

2.3.3. PHP

PHP je skriptovací jazyk používající se jako prostředek dynamizace webových stránek na straně serveru. Jeho vznik je datován do roku 1995, kdy byly položeny jeho základy. Výhodou PHP je jeho snadné pochopení a možnost práce s malými znalostmi. To ale neznamená, že pomocí PHP nejdou dělat i velké věci.

PHP umí pracovat s velkým počtem databázových systémů, textovými soubory, umí zpracovávat fotografie, pdf, nastavovat vlastnosti serveru a spoustu jiných věcí, takže programátor není technologicky v podstatě omezen.

Pro vývojáře AJAXových aplikací je klíčové, že PHP je schopné vytvářet XML dokumenty nebo případně i data ve formátu JSON.

Jeho zdrojové kódy jsou uloženy na serveru, takže se k nim nikdo zvenčí nedostane.

PHP obvykle pracuje na serveru Apache a jeden z důvodů, proč PHP používám je jeho velká rozšířenost a menší náročnost.

2.3.4. CSS

Cascading Sytle Sheets neboli česky Kaskádové styly jsou navrženy jako možnost grafické úpravy stránek. První návrh je z roku 1994 a dnešní používaná verze CSS 2 je o čtyři roky starší. Blíží se však doba, kdy nastoupí verze 3, která nemá dnes ještě podporu v prohlížečích, avšak měla by umožňovat některá nová a dokonalejší grafická vyjádření.

Hlavní cíl je jednoduše naformátovat (X)HTML a XML data tak, aby grafické vyjádření bylo co nejvíce oddělené od struktury dat. Dále rozšiřuje grafické možnosti, které nabízí přímo (X)HTML.

Pomocí CSS je možné jednoduše nadefinovat jednotný vzhled určitých elementů ve stránce nebo naopak vytvořit formát pouze pro jeden vybraný element. Jeden styl můžeme použít pro libovolný počet elementů ve stránce, což má své nesporné výhody:

- Ubude nám velké množství kódu, kdy bychom si jinak museli popisovat grafiku každého elementu na stránce zvlášť.
- Snadné budou editace pouze na jednom místě v kódu pro velký počet zahrnutých elementů ve stylu

Kaskádový styl stránky se skládá z pravidel, které se zapisují ve tvaru:

```
nazev_elementu{vlastnost: hodnota;}
```

Název elementu je konkrétní název, např. div. V tom případě všechny divy na stránce budou mít konkrétní vlastnost nastavenou na určitou zadanou hodnotu. Každé pravidlo může mít libovolný počet vlastností.

```
div{color: white;width: 100px;}
```

Dále je možné specifikovat nějaké námi vytvořené pravidlo nějakého určitému elementu. To se dělá pomocí atributů class nebo id, které přidáme do elementu. Vložení Kaskádového stylu provádíme podobně jako u JavaScriptu. Buď přímo do kódu pomocí tagu style:

```
<style type="text/css">body{color: blue}</style>
```

nebo jako externí soubor pomocí tagu link:

```
<link rel="stylesheet" type="text/css" href="styl.css" />
```

anebo přímo do elementu pomocí atributu style:

```
<div style="color: white;">nějaký text</div>
```

2.3.5. XML

XML (Extensible Markup Language) je dnes opravdu velký pojem, se kterým se můžeme setkat nejen v oblasti tvoření internetových prezentací, ale lze ho výhodně nasadit při popisu a přenosu dat v podstatě ve všech oblastech informačních technologií. Proto je hojně využíván i technikou AJAX. Hlavně z toho důvodu, že je dobře standardizován a se správně navrženým XML dokumentem lze snadno manipulovat pomocí JavaScriptu nebo nějaké dnešní serverové technologie, pro kterou se při návrhu aplikace na AJAXu rozhodneme. V mém případě PHP.

XML je složené z tzv. elementů, které jsou podobné již známým HTML tagům. Rozdíl je v tom, že elementy a atributy XML si můžeme libovolně vytvářet a pojmenovávat přesně pro popis dat, které budeme v XML uchovávat.

Obecný zápis elementu:

```
<element atribut="hodnota _atributu">
      Data, která tento element zaštiťuje
</element>
```

Správně napsaný XML dokument musí splňovat tato pravidla, z nichž jsou některá shodná se správně napsaným XHTML kódem, který z XML vychází:

- Na začátku zdrojového kódu by měla být XML deklarace s možným používaným kódováním (defaultně se uvažuje UTF-8)

```
<?xml version="1.0" encoding="utf-8"?>
```

- Celý další dokument je uzavřen v tzv. kořenovém elementu (root elementu)
- Všechny elementy i atributy musí být malými písmeny
- Všechny hodnoty atributů musí být v XML v uvozovkách.
- Všechny XML elementy musí být párové. Při použití prázdného elementu se musí tag ukončit lomítkem, např. <prazdny />
- Elementy se nesmí nikdy křížit.

Ukázka navrženého XML pro uchování dat o zaměstnanci:

```
<?xml version="1.0"?>
<vsichni_zamestnanci>
   <zamestnanec identifikator="1">
       <jmeno pohlavi="muz">Jan Novák</jmeno>
       <mesto>České Budějovice</mesto>
       <adresa>Budějovická 155</adresa>
       <psc>37001</psc>
    </zamestnanec>
</vsichni_zamestnanci>
```

Z takového kódu jsme nyní schopni získat údaje o Janu Novákovi .

2.3.6. JSON

JSON (JavaScript Object Notation) je odlehčený formát pro výměnu dat. Je to alternativa k XML, jehož použití v AJAXu je na místě.

Je jednoduše čitelný i zapisovatelný člověkem a snadno analyzovatelný i generovatelný strojově. Je založen na podmnožině programovacího jazyka JavaScript.

JSON je textový, na jazyce zcela nezávislý formát, využívající však konvence dobře známé programátorům jazyků rodiny C. Díky tomu je JSON pro výměnu dat opravdu ideálním jazykem [8].

JSON je založen na dvou strukturách:

- Kolekce párů název/hodnota. Ta bývá v rozličných jazycích realizována jako objekt, záznam, struktura, slovník, hash tabulka, klíčový seznam nebo asociativní pole.
- Tříděný seznam hodnot. Ten je ve většině jazyků realizován jako pole, vektor, seznam nebo posloupnost.

Já ve svém programu JSON nepoužívám, nicméně jsem ho zkoušel a musím uznat, že v některých případech mi přišel jako vhodné řešení, které je schopné XML konkurovat. Někdo říká, že pro použití v AJAXu je JSON výhodnější, protože jeho parsování je rychlejší. Co se týče PHP, tam je situace taková, že JSON knihovny nejsou standardně zavedeny, ale dají se dodat. Servery s novějšími verzemi PHP většinou již podporu formátu JSON zahrnují.

Nakonec ještě příklad zápisu stejné struktury dat nejprve pomocí XML a poté JSON. Můžete si vybrat.

```
<zamestnanec>
   <jmeno>Jan Novák</jmeno>
   <mesto>České Budějovice</mesto>
   <adresa>Budějovická 155</adresa>
   <psc>37001</psc>
</zamestnanec>

{"zamestnanec": {
   "jmeno ": "Jan Novák",
   "mesto": "České Budějovice",
   "adresa ": "Budějovická 155",
   "psc ": "37001",
}}
```

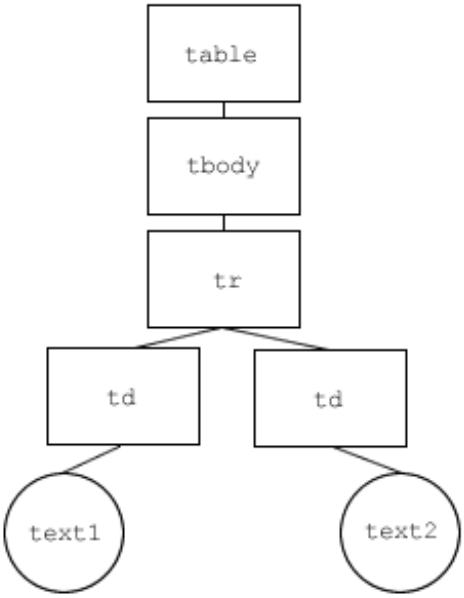
2.3.7. DOM

W3C DOM je specifikace pro přístup k obsahu a struktuře dokumentu, který je nezávislý na platformě. Je to způsob jak běžně reprezentovat HTML a XML dokument a poté s ním manipulovat.

Jedná se o objektový model, na nějž můžeme pohlížet jako na stromovou strukturu. Např. i na vyjádření klasické tabulky pomocí HTML:

```
<table>
   <tbody>
      <tr>
         <td>Text1</td>
         <td>Text2</td>
      </tr>
   </tbody>
<table>
```

Na tuto strukturu můžeme pohlížet jako na následující obrázek.



Obrázek I. Příklad pohledu na DOM dokument

Takto se dá nahlížet na celou stránku se všemi jejími prvky a vlastnostmi, kde každý prvek má nějakého rodiče, případně své potomky a sourozence. To je pro programátora v JavaScriptu a AJAXu velmi důležité si uvědomit, protože s takovými strukturami pracuje doslova na každém "kroku". Bez DOMu by nejzajímavější možnosti AJAXu nebyly možné, protože pomocí JavaScriptu nemusíme tímto modelem pouze procházet, ale můžeme i elegantně upravovat jeho obsah.

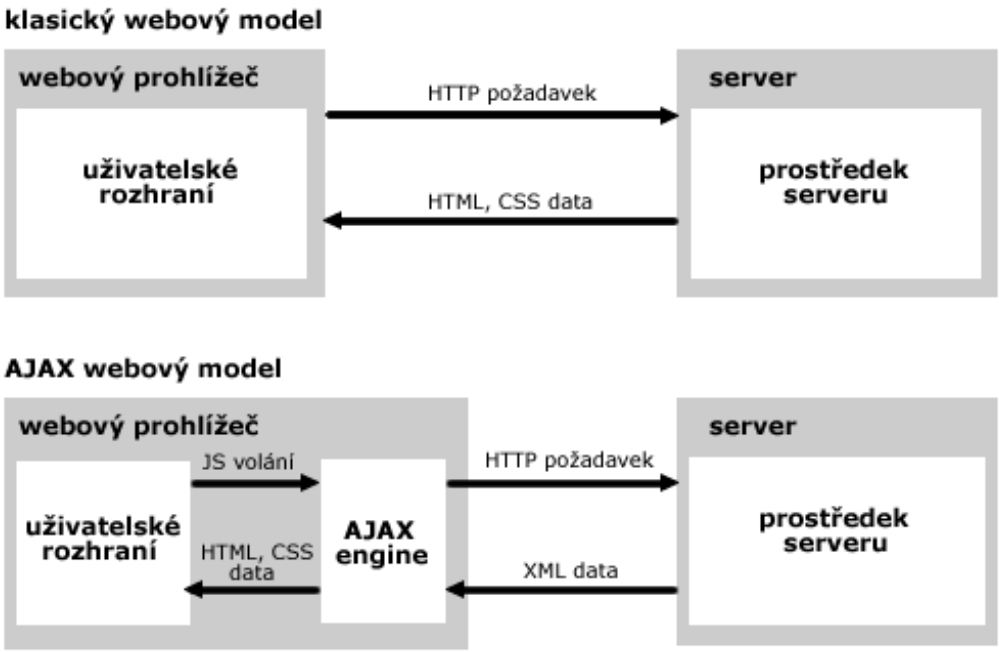
2.4. AJAX vs. klasický postup

AJAX umožňuje vytvářet nové funkce. Proto je také jasné, že se model, na kterém AJAX pracuje, bude od klasického modelu nějak odlišovat.

Zatímco klasický postup pracuje tak, že uživatel skrz nějaké webové rozhraní vyvolá nějakou akci, např. Kliknutím na odkaz. Na server se odešle HTTP požadavek a server zpracuje odpověď. Tedy může načíst data z

nějakého úložiště a vytvoří celý nový aktuální zdrojový kód stránky, který je poslán webovému prohlížeči a ten se postará o jeho grafické vyjádření. (viz. obr.II).

U AJAXového modelu existuje ještě mezi serverem a prohlížečem jakýsi mezičlánek, jakýsi AJAX engine vytvořený pomocí JavaScriptu. Uživatel komunikuje se serverem právě přes tento engine. Tedy nějakým voláním (např. událost "onclick" v JavaScriptu) kontaktuje AJAX engine, který sám vytvoří HTTP požadavek, provede jakési neviditelné volání na pozadí prohlížeče. Server data klasicky vytvoří, ovšem pouze ty, o které si uživatel svým chováním řekl. Zatímco uživatel může dále pokračovat v práci, jsou data v nějakém vhodném formátu (např. v XML) opět předána AJAX engine, který je zpracuje, tedy vytvoří jejich HTML podobu. Nakonec je engine pomocí JavaScriptu zobrazí na nějaké určité místo ve stránce. (viz. Obrázek II).



Obrázek II. Blokové schema klasického modelu a modelu s RIA AJAX

2.5. Důležité dovednosti v JavaScriptu pro práci s AJAXem

2.5.1. Objekt XMLHttpRequest

Dá se říci, že právě objekt XMLHttpRequest je základním kamenem AJAXu. Přes tento objekt je možné vytvářet asynchronní požadavky na server.

2.5.1.1. Náhled do Historie XMLHttpRequest

Poprvé se XMLHttpRequest objevil v roce 1999 v prohlížeči Internet Explorer 5 od firmy Microsoft jako objekt ActiveX. To, že s ním programátoři začali spolupracovat, způsobilo až jeho zavedení v Mozzile 1.0. V dnešní době je implementován ve všech moderních webových prohlížečích jako jsou Mozzila, Firefox, Opera, Safari, Netscape Navigator, Internet Explorer či Konqueror, i když není W3C standardem. Taková podpora v prohlížečích XMLHttpRequest znamená možnost nabídnout AJAX drtivé většině dnešních uživatelů internetu, kteří vesměs tyto prohlížeče využívají.

2.5.1.2. Vytvoření instance XMLHttpRequest

Kromě Internet Exploreru 6 můžeme vytvořit přímo instanci objektu. To ale neznamená, že v IE6 bychom byli o AJAX ochuzeni, jen je třeba vyvolat XMLHttpRequest jinou cestou.

Pro vytvoření instance objektu vytvoříme krátký JavaScriptový kód, který by nám měl zaručit vytvoření instance ve všech prohlížečích, ve kterých je toto možné, včetně IE6.

Pro IE6 je objekt pořád komponentou ActiveX, proto jeho instanci vytvoříme následujícím kódem:

```
xmlHttp = new ActiveXObject('Microsoft.XMLHTTP');
```

U ostaních prohlížečů:

```
xmlHttp = new XMLHttpRequest();
```

Tedy i s logikou a zabalený do funkce bude náš kód vypadat takto

```
function createXmlHttpRequest(){
      var xmlHttp;
      if(window.ActiveXObject){
            xmlHttp = new
      ActiveXObject('Microsoft.XMLHTTP');
       }
      else if(window.XMLHttpRequest){
            xmlHttp = new XMLHttpRequest();
      }
      if(!xmlHttp)alert("Nepodařilo se zavést objekt.");
      else return xmlHttp;
}
```

Není to tedy složitá záležitost. `Window.ActiveXObject` vrací objekt nebo null, což je vyhodnoceno pomocí podmínky if. Pokud je podmínka vyhodnocena jako `false`, je v další podmínce testováno, zda `window.XMLHttpRequest` vrací objekt.

`Microsoft.XMLHTTP` je však objekt z nejstarší knihovny ActiveX, která do dnešních dnů čítá množství různých verzí. Proto si předchozí skript ještě obohatíme o logiku, která zaručí, že vyvoláme vždy nejnovější dostupnou verzi objektu.

Celá podmínka pro IE6 bude rozšířena do následující podoby:

```
if(window.ActiveXObject){
      var XmlHttpVersions = new Array(
                        'MSXML2.XMLHTTP.6.0',
                        'MSXML2.XMLHTTP.5.0',
                        'MSXML2.XMLHTTP.4.0',
                        'MSXML2.XMLHTTP.3.0',
                        'MSXML2.XMLHTTP',
                        'Microsoft.XMLHTTP');

      for (var i = 0; i < XmlHttpVersions.length;i++){
            try{
            xmlHttp=new ActiveXObject(XmlHttpVersions[i]);
            }
            catch (e) {}
      }
}
```

Vytvoříme si pole `XmlHttpVersion` s jednotlivými verzemi objektu, jež v dalším kroku projdeme v cyklu, který nejprve zkoumá, zda lze vytvořit nejnovější verzi `MSXML2.XMLHTTP.6.0` a při každém neúspěchu je testována pomocí bloku `try/catch` verze o řád nižší. Dostáváme se tedy až k naší nejstarší verzi `Microsoft.XMLHTTP`. Zbytek kódu zůstane z minulého příkladu beze změn.

Nyní již můžeme vytvořit slíbenou instanci. Nejprve si vytvoříme globální proměnnou s názvem `xmlHttp` a zavoláním naší funkce do ní instanci umístíme.

```
var xmlHttp; // deklaruje se proměnná s názvem xmlHttp
xmlHttp =  createXmlHttpRequest();
```

2.5.1.3. Metody a atributy XMLHttpRequest

Ať už vytvoříme instanci XMLHttpRequest pro libovolný prohlížeč a verzi, je možné používat jeho metody a atributy, které vyjadřují následující tabulky.

Metoda	**Popis**
abort()	Přeruší požadavek
getAllResponseHeaders(“hlavička”)	Vrátí všechny hlavičky požadavku ve formě klíč/hodnota
getResponseHeader(“metoda”, ”url”)	Vrátí hodnotu zadané hlavičky ve formě řetězce
open(“metoda”, “url”,”async”)	Nastaví parametry volání serveru. Argument *metoda* může nabývat hodnot GET, POST, PUT. Argument *url* představuje požadované URL. Dále existují další tři nepovinné argumenty, z nichž je přro nás podtatné nastavení možnosti asynchronního volání.
send(obsah)	Odešle požadavek serveru
setRequestHeader(“hlavička”, “hodnota”)	Nastaví zadanou hodnotu nějaké hlavičce.

Tabulka I. Metody objektu XMLHttpRequest

Atribut	**Popis**
onreadystatechange	Ukazatel na obslužný kód, který je spuštěn při každé změně interního stavu objektu
readyState	Stav požadavku. Nabývá pěti možných hodnot: 0 – neinicializovaný 1 – zavádí se 2 – je zaveden 3 – přechodný 4 – dokončeno
responseText	Odpověď serveru ve formě řetězce (holý text)
responseXML	Odpověď serveru ve formě XML. Objekt může být dále zpracováván jako DOM objekt

status	Stavový kód získaný ze serveru. 200 – operace byla úspěšně provedena 404 – prostředek nelze nalézt atd.
statusText	Textová verze stavového kódu

Tabulka II. Atributy objektu XMLHttpRequest

Metody, které budeme požívat nejčastěji jsou `open` a `send`. Budeme je volat při každém požadavku na server. Požadavek pracuje asynchronně, jestliže třetí parametr funkce `open` nastavíme na true, což budeme dělat v podstatě pokaždé, protože o asynchchronním chování požadavku AJAX je. V takovém případě je nutné ještě před voláním funkce send nastavit událost onreadystatechange na nějakou funkci, která se provede při změně stavu požadavku.

2.5.2. DOM a JavaScript

Bez DOM by JavaScript nebyl schopen jednoduše přistupovat k elementům HTML a XML. Každý element těchto dokumentů je částí DOM. Pro přístup k DOM můžeme použít v podstatě jakýkoli skriptovací jazyk, my však děláme aplikaci pomocí AJAXu, proto budeme nejčastěji přistupovat k DOM pomocí JavaScriptu.

Pro zpracování XML a XHTML dokumentů budeme potřebovat určité metody a atributy, které můžeme přes DOM používat.

Metoda	**Popis**
getElementById(‘id’)	Vrací dokument, který je identifikován pomocí atribitu id
getElementByTaName(‘název’)	Vrací pole potomků element identifikovaného podle názvu značky elementu
hasChildNodes()	Vrací Boolean hodnotu, která udává, jestli má element nějaké potomky

getAttribute(‘název’)	Vrací hodnotu atributu se jménem název danného elementu

Tabulka III. Metody pro práci s DOM

Atribut	**Popis**
childNodes	Vrací pole potomků danného elementu
firstChild	Vrací prvního přímého potomka daného elementu
lastChild	Vrací posledního přímého potomka daného elementu
nextSibling	Vrací element následující za daným elementem
nodeValue	Specifikuje atribut pro čtení/zápis reprezentující hodnotu elementu
parentNode	Vrací rodičovský uzel elementu
previousSibling	Vrací element předcházející danný element

Tabulka IV. Atributy pro práci s DOM

Asi nejčastěji přistupujeme k nějakému elementu pomocí objektu `document` a metody `getElementById`. V praxi to vypadá tak, že máme nějaký element například div, který má nastavený atribut id.

```
<div id="menu">nějaké menu</div>
```

Pomocí JavaScript DOM je možné tento element získat a naplnit s ním proměnnou menu takto:

```
var menu = document.getElementById('menu');
```

Druhá častá operace je načtení elementů stejného významu do pole a poté je sekvenčně zpracovat, např. pomocí nějakého cyklu. Například získání všech potomků option, jejichž rodič je element select s určitým id:

```
<select id="level">
    <option>první</option>
    <option>druhý</option>
</select>
```

Provedeme takto:

```
var level = document.getElementById('level');
var levely = level.getElementByTagName('option');
```

S takto získanými daty můžeme libovolně nakládat, jak bude také vidět dále v textu. Navíc můžeme pomocí DOMu elementy, se kterými jsme doposud pracovali, vytvářet či odebírat. Pro tyto možnosti existují také specifické prostředky.

Metoda	**Popis**
document.createElement(název)	Vytváří element specifikovaný jeho argumentem
document.createTextNode(text)	Vytvoří uzel obsahující text
<element>.appendChild(uzel)	Přidá zadaný uzel do seznamu potomků danného elementu
<element>.getAttribute(název) <element>.setAttribute(název,hodnota)	Tyto metody slouží pro získání resp. nastavení hodnoty atributu elementu specifikovaného argumentem název
<element>.removeAttribute(název)	Odstraní atribut zadaný argumentem název z elementu
<element>.removeChild(uzel)	Odstraní uzel zadaný argumentem uzel ze seznamu potomků elementu
<element>.raplaceChil(novýUzel,starýUzel)	Nahradí uzel zadaný argumentem nový uzel za uzel zadaný argumentem starýUzel
<element>.hasChildNodes()	Vrací hodnotu Boolean, která určuje, zda má element nějaké potomky

Tabulka V Prostředky pro změny struktury DOM

Pomocí konstrukcí v tabulce můžeme snadno vytvořit například celý select s potomky option a umístit ho např. do divu s nějkým id.

```
var selectElm = document.createElement("select");

var optionElm = document.createElement("option");
var optionText = document.createTextNode("první");
optionElm.appendChild(optionText);
selectElm.appendChild(optionElm);

optionElm = document.createElement("option");
optionText = document.createTextNode("druhý");
optionElm.appendChild(optionText);
selectElm.appendChild(optionElm);

optionElm = document.createElement("option");
optionText = document.createTextNode("třetí");
optionElm.appendChild(optionText);
selectElm.appendChild(optionElm);

document.getElementById("selectDiv").appendChild(selectElm);
```

Nejprve jsem si vytvořil pomocí `createElement` element select, který je zatím v paměti. Poté stejnou cestou element option. Pomocí `createTextNode` jeho text (“první”). Tento text jsem prostřednictvím `appendChild` přidal elementu option, který jsem stejnou cestou přidal do selectu. Takto jsem pokračoval pro zbylé dva elementy option. Nakonec jsem select přidal opět pomocí `appendChild` do předpokládaného existujícího divu s id o hodnotě “selectDiv”. V tuto chvíli je již můj kód vytvořený pomocí DOM JavaScriptu vidět a lze ho používat stejně, jako bychom ho vytvořili přímo v HTML do statického kódu.

Na tomto příkladu právě často AJAX staví a XML data získaná ze serveru zpracovává tímto způsobem.

2.5.3. Použití atributu innerHTML

Poprvé se tento atribut objevil u Internet Exploreru, ale dnes je již rozšířen ve většině prohlížečích. I přesto, že se nejedná o standardizovaný atribut, je v AJAXu hojně využíván. Pomocí tohoto prostředku můžeme některým prvkům na stránce vytvářet buď čistě textovou náplň, anebo náplň HTML, která bude samozřejmě interpretována a zobrazena v podobě, kterou má vyjařovat.

Toto je jeden z aspektů, jak vytvořit funkčnost, kdy se změní obsah určitého místa na stránce a ostatní zůstane beze změn.

Klasickým příkladem bývá naplnění elementu div. Budeme zde využívat dovedností z minulé kapitoly. Předpokládejme prázdný div:

```
<div id="mujDiv"></div>
```

Pomocí innerHTML můžeme vytvořit jeho libovolný obsah takto:

```
var html = "<strong>hello world</strong>";
html += "<span id="comp">hello computer</span>";
document.getElementById("mujDiv").innerHTML = html;
```

Takto dynamicky vytvořený obsah divu je srovnatelný se statichým zápisem:

```
<div id="mujDiv">
   <strong>hello world</strong>
   <span id="comp">hello computer</span>
</div>
```

2.5.4. Použití CSS stylů JavaScriptem

Nejedná se přímo o dovednost, která nějak ovlivňuje způsob provádění AJAXových funkcí. Avšak je to věc, která je s AJAXem a RIA aplikací úzce

spjata. Proto jen v rychlosti.

CSS styly jdou snadno nastavovat pomocí DOM JavaScriptu i poté, co je stránka již načtena. Jakémukoli elementu se dá nastavit nějaká vlastnost takto:

```
document.getElementById("nejakyDiv").style.color = "white";
```

nebo přiřazením vytvořené CSS třídy takto:

```
document.getElementById("nejakyDiv").className = "CSSTřída";
```

Tato dovednost často hraje roli v aktualizaci různých míst na stránce, kdy potřebujeme např. nějakou část skrýt, ukázat, nebo nějak graficky pozměnit. Funkce jako je např. “Drag And Drop” by se bez této dovednosti určitě neobešly.

2.6. První program v AJAXu

2.6.1. XHTML dokument

Při návrhu AJAXových funkcí je dobré si nejprve vytvořit kostru (X)HTML dokumentu, se kterou budeme pomocí AJAXu manipulovat. V prvním programu jde o to si pouze vyzkoušet, jak nám bude AJAX fungovat, proto volím jen jednoduchý XHTML dokument s jedním divem a jedním odkazem.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="cs"
lang="cs">
```

```
<head>
        <title>První program v AJAXu</title>
        <script type="text/javascript" src="ajax.js">
        </script>
</head>

<body>
        <div>
              <span onclick="sendRequest();">
                    Textový odkaz
              </span>
        </div>
        <div id="vysledek">
              Zde bude zobrazen text ze serveru
        </div>
</body>
</html>
```

Toto nám pro začátek bude stačit. Předpokládáme zde existenci souboru ajax.js, v němž bude zdrojový kód JavaScriptu. Zatím v něm předpokládáme pouze již vysvětlovanou funkci `createXmlHttpRequest` a globální proměnnou s vytvořenou instancí objektu XMLHttpRequest.

Odkaz jsem záměrně nevytvořil pomocí elementu a, ale pomocí elementu span, který se používá pro formát nějaké části textu. Funkci, která vyvolá akci po kliknutí na text v tomto elementu, zajišťuje přítomnost události onclick, která zavolá funkci `sendRequest`. Tu si teprve vytvoříme. Dále si všimněme, že element div má atribut id s hodnotou výsledek. To je nutné k tomu, abychom měli jednoznačný ukazatel na místo, kam budeme vkládat získaná data ze serveru.

2.6.2. Funkce sendRequest pro odeslání požadavku

Nyní si vytvoříme funkci sendRequest:

```
function sendRequest(){

   if (xmlHttp.readyState == 4 || xmlHttp.readyState == 0){
         try{
               var url = "./file.php";
               xmlHttp.open("GET", url, true);
               xmlHttp.onreadystatechange =
                     function(){handleRequestState();};
               xmlHttp.send(null);
         }
         catch(e){}
   }
   else setTimeout('sendRequest()',1000);
}
```

Na první pohled se může zdát funkce poněkud nečitelná, ale není to nic složitého. Nejprve je v podmínce testováno, zda není objekt xmlHttpRequest zaneprázdněn. V případě, že ano, je nastaven časovač, který nám v další sekundě spustí metodu znovu.

V případě, že je to možné, vytvoří se řetězec, který obsahuje adresu php skriptu. Ten bude na serveru vykonán. Pomocí metody `open` říkáme, že se má použít metoda GET pro přenos dat, náš soubor (file.php) a pracovat se bude v asynchronním režimu. Dále nastavíme události `onreadystatechange` metodu, která zpracuje odpověď. V našem případě je to `handleRequestState`, kterou si za chvíli vytvoříme. A nakonec pošleme data serveru pomocí funkce `send`, jež má parametr `null`, což je běžné nastavení v případě, že používáme přenos typu GET.

2.6.3. PHP skript

Podíváme se na soubor file.php:

```
<?php
   echo "<strong>Text získaný ze serveru</strong>";
?>
```

Toto je vše, co jsem do něj zatím umístil. Když ho tedy server zpracuje, jediné co provede je, že vypíše tučně text pomocí PHP funkce `echo`. Na první pohled by se mohlo zdát, že je to poněkud chudé, ale uvědomme si, že v tomto souboru může být i velmi propracovaná logika a skript nám může tisknout data, která mohou představovat nejrůznější hodnoty, získané navíc z nejrůznějších zdrojů (soubory, databáze, internet …).

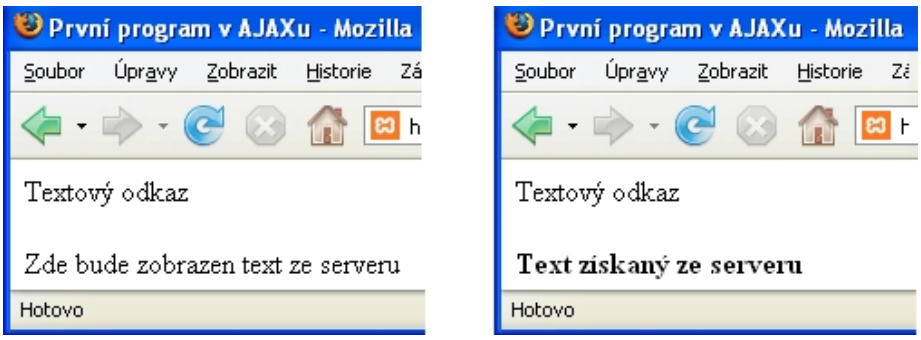
2.6.4. Funkce handleRequestState pro zpracování dat

Nyní si vytvoříme poslední funkci a to již slíbenou handleRequestState do souboru ajax.js:

```
function handleRequestState(){

   if (xmlHttp.readyState == 4){
         if (xmlHttp.status == 200){
               document.getElementById('vysledek').innerHTML=
                     xmlHttp.responseText;
               }
   }
}
```

Funkce obsahuje podmínky, které testují, zda je transakce dokončena a že je dokončena úspěšně. Jestliže ano, je pomocí DOM modelu a JavaScriptu

umístěn text ze souboru file.php, který je obsažen ve vlastnosti `responseText`. Do patřičného divu je přiřazen díky vlastnosti `innerHTML` každého divu. Na náš div se odkazujeme pomocí `document.getElementById('vysledek')`, kde výsledek je hodnota atributu divu pro zobrazení dat.



Před kliknutím na Textový odkaz Po kliknutí na Textový odkaz

Obrázek III. Stavy prvního programu

A toto je pro začátek vše. Nejprve je třeba si vše pořádně promyslet, protože ze začátku není úplně snadné se s jiným přístupem vyrovnat. Není to však nic tak těžkého.

Tento program je opravdu jen základ. Pracuje přímo s holým textem, který přijímáme ze serveru a přímo ho zobrazuje. Toto se v praxi používá jen málo, v podstatě pouze u malých akcí. Dále si ho rozšíříme o funkčnost, která bude pracovat s XML dokumentem, jak bývá nejčastější. Ale už zde můžeme vidět, že to co proběhlo bylo opravdu jaksi na pozadí prohlížeče a text se objevil v divu bez refreshe prohlížeče.

2.7. Vylepšený program pomocí XML

2.7.1. XHTML dokument

V tomto vylepšeném programu budeme používat stejnou kostru XHTML

dokumentu. Pouze přidáme ještě jeden textový odkaz vytvořený opět pomocí elementu span.

```
<span onclick="sendRequest(1);">Textový odkaz1</span>
<span onclick="sendRequest(2);">Textový odkaz2</span>
```

2.7.2. Funkce sendRequest pro odeslání požadavku

Další změna je přidání jednoho parametru funkci `sendRequest`, jehož význam si popíšeme dále.

Část funkce `sendRequest` změníme do následující podoby:

```
function sendRequest(ident){

   if (xmlHttp.readyState == 4 || xmlHttp.readyState == 0){
         try{
               var url = "./file.php?ident=" + ident;
               //tady již bude vše jako minule
               .
               .
}
```

Parametr `ident` nám reprezentuje proměnnou, kterou budeme pomocí metody GET zasílat PHP skriptu. Její pomocí bude možné vytvořit rozhodovací logiku o tom jaká data nám server pošle zpět. Dělá se to stejně jako klasickou cestou přes klasický odkaz. Tedy rozšířením url, kde za adresou skriptu a znakem "?" následuje jméno proměnné pro PHP, za kterou znak "=" určuje jaká bude její hodnota. Tuto hodnotu získáme z parametru funkce `sendRequest`. Tedy při pohledu na naše odkazy bude při volání serveru buď 1, nebo 2.

2.7.3. PHP skript

Podstatnější změny zaznamená soubor file.php:

```
<?php
   header('Content-Type: text/xml');

   if($_GET["ident"] == 1)$data = "České Budějovice";
   elseif($_GET["ident"] == 2)$data = "Třeboň";

   $dom = new DOMDocument();
   $rootElement = $dom->createElement("data");

   $dom->appendChild($rootElement);

   $dataElement = $dom->createElement("mesto");
   $value = $dom->createTextNode($data);
   $dataElement->appendChild($value);

   $rootElement->appendChild($dataElement);

   $xmlString = $dom->saveXML();
   echo $xmlString;
?>
```

Když vytváříme pomocí PHP XML dokument, musíme si vybrat ze dvou možností. Můžeme buď vytvářet XML ručně, tedy spojováním řetězců do výsledné podoby, nebo můžeme využít DOM modelu PHP, který funguje stejně jako u JavaScriptu, jen s rozdílem používání syntaxí PHP. Budu používat druhou možnost, která má nespornou výhodu v tom, že vzniklý kód bude správně napsán a proto se nemusím bát toho, že např. zapomenu někde ukončovací znak elementu apod. Tuto výhodu doceníme při nějakém rozsáhlejším XML, kdy se na DOM budeme moci spolehnout.

Prvním příkazem nastavím Content-Type souboru na hodnotu text/xml, což říká, že celý skript představuje XML dokument.

K proměnné `ident`, kterou jsme poslali asynchronním požadavkem, zde můžeme přistupovat pomocí PHP vyjádření superglobálních proměnných typu GET. Tedy `$_GET['ident']`.

Zvolil jsem jen jednoduché podmínky, na kterých je dobře vidět, že skript porovnává obsah proměnné s hodnotami 1 a 2. V případě rovnosti jedné z hodnot naplní skript lokální proměnnou `$data` jedním ze dvou měst. Nepoužívám tedy zatím žádné načítání z nějakého databázového zdroje, ale pouze přednastavené hodnoty.

Dalším krokem je vytvoření instance objektu DOM pomocí `$dom = new DOMDocument();` Přes tuto proměnnou je možné využívat metody objektu, z nichž jako první přijde na řadu vytvoření kořenového elementu data pomocí `createElement`. Poté stejným způsobem vyrobíme element mesto, který později umístíme do kořenového elementu. Nejprve však metodou `createTextNode` zhotovíme textový obsah, který pomocí `appendChild` umístíme do elementu vysledek. Celý element mesto opět přes `appendChild` přidáme do kořenového elementu data.

Nakonec vytvořím celý dokument v podobě řetězce pomocí DOM funkce `saveXML` a řetězec vytisknu, což mi zajistí, že ze souboru file.php vznikne XML dokument o této struktuře:

```
<?xml version="1.0"?>
<data>
   <mesto>-Vybrané město-</mesto>
</data>
```

2.7.4. Funkce handleRequestState pro zpracování dat

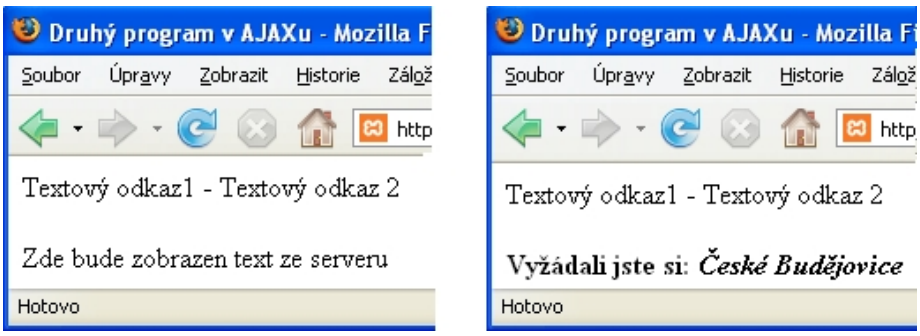
Funkce `handleReaquestState` prošla také změnou. Už nezobrazuje prostý text, který přijde ze serveru, ale text ze serveru nejprve zpracuje a získaná data naformátuje do XHTML podoby.

```
function handleRequestState(){

   if (xmlHttp.readyState == 4){
      if (xmlHttp.status == 200){

         var xmlResponse = xmlHttp.responseXML;
         var root = xmlResponse.documentElement;
         var cities = root.getElementsByTagName("mesto");
         var city = cities.item(0).firstChild.data;

         var html = "Vyžádali jste si: <em>"+city+"</em>";
         html = "<strong>" +html+ "</strong>";

         document.getElementById('vysledek').innerHTML =
         html;
      }
   }
}
```

Ověřování, zda je transakce úspěšně dokončena pořád zůstává, ale dále musíme z proměnné `xmlHttp` získat příchozí XML data ze serveru pomocí vlastnosti `responseXML`, ve které jsou obsaženy. Uložíme je do proměnné `xmlResponse`. Nyní již můžeme využít naše znalosti JavaScript DOMu. Nejprve načteme do proměnné `root` kořenový element pomocí `xmlResponse.documentElement`. Dále získáme všechny elementy mesto do pole, které představuje proměnná cities `getElementsByTagName`. Protože víme, že přijde pouze jedno město (díky logice na serveru není možné získat

více elementů mesto), můžeme použít hned první a jedinou položku v poli. Provedeme to např. takto: `item(0).firstChild.data`. Takto získáme textovou hodnotu z elementu město.

Nakonec vytvoříme proměnnou `html`, ve které vytvoříme řetězením XHTML obsah, který pomocí vlastnosti `innerHTML` umístíme opět do divu s patřičným id.



Před kliknutím na text Po kliknutí na první text

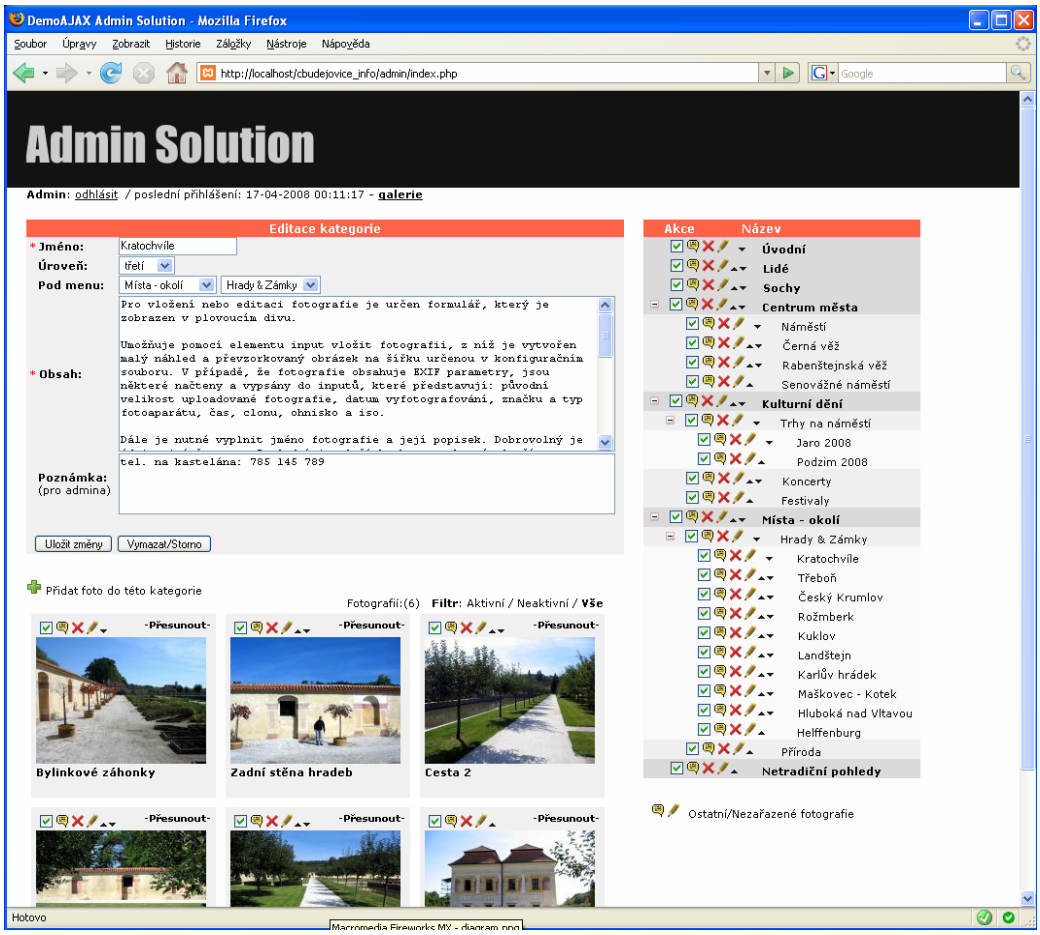
Obrázek IV. Stavy duhého programu

Alternativním řešením by bylo nevytvářet výsledný kód řetězením a vložením pomocí `innerHTML`, ale použít již vysvětlovaný postup pomocí DOM JavaScriptu a do výsledného divu vkládat obsah pomocí metody `appendChild`. Toto je vždy na nás, co je v danou chvíli výhodnější. Na těchto ukázkách budu dále stavět při popisu praktické aplikace.

3. Realizace praktické aplikace

3.1. Představení aplikace

Jako praktickou ukázkovou část jsem se rozhodl vytvořit aplikaci, pomocí níž bude možné vytvářet libovolně rozsáhlou strukturu článků a fotogalerií. Tato data budu ukládat do databáze a budou zpětně editovatelná.



Obrázek V. Náhled do aplikace

Administrační rozhraní je navržené přesně podle potřeb, které jsem si

vymyslel a budou přesně splňovat moje nároky, jaké od takovéhoto systému vyžaduji. Myšlenka vzikla ve spolupráci se známým fotografem na základě mnohokrát opravovaných nákresů, kde jsem se snažil od začátku vymyslet nejlepší funkční řešení věci.

Všechna funkčnost od nejmenší akce po ty rozsáhlejší je řešena pouze pomocí RIA AJAX a při práci v programu nedochází k žádné aktualizaci stránky. Vše se děje pomocí asynchronních požadavků na server.

3.1.1. Význam a rozvržení částí stránky

3.1.1.1. Menu

V pravé části stránky je umístěné menu (viz. Obrázek V). Toto menu je tvořené dynamicky mým programem. Menu může mít hloubku jednu až tři úrovně. Každá z položek jakékoli úrovně v menu může být nějaký článek, fotogalerie nebo obě dvě možnosti najednou.

Položky menu nabízejí přímo možnosti pro:

- *Schování resp. zobrazení existujících podkategorií* – V případě, že se jedná o úrovně první a druhá, je možné si schovat všechny podúrovně příslušné položky. Možnost slouží pro lepší orientaci ve struktuře menu a je zobrazována pouze v případě, existují- li nějaké podkategorie
- *Aktivace resp. deaktivace* – Tím je myšleno zobrazení na výstupu. Například při ještě nevytvořeném obsahu bude vidět položka pouze v tomto adminu.
- *Zobrazení detailu položky* – Po kliknutí na tento odkaz se zobrazí div s deatilem textu a fotografií obsažených v položce.
- *Vymazání položky ze seznamu* – Případné fotografie smazány

nejsou, pouze jsou přesunuty do složky Ostatní/Nezařazené fotografie.

-  *Editace položky* – Údaje týkající se položky jsou připravené pro jejich editaci. Jsou načtené do formuláře, který je vytvořil
- *Posouvání položky ve směru šipky* – Posouvání je možné pouze v rámci své kategorie, slouží jako možnost změny pořadí položek

Poslední položka umístěná dole pod menu je stálá a nedá se z menu odstranit. Slouží jako odkládací prostor pro nezařazené fotografie nebo jako místo pro fotografie, jejichž menu bylo odstraněno a nebylo prázdné.

3.1.1.2. Formulář pro práci s položkami menu

Vlevo na stránce je umístěn formulář (viz. Obrázek V), přes který se dají vkládat nebo editovat položky v menu.

Jeho prvním prvkem je input, který představuje jméno položky. Vyplnění je povinné, bez něj nejde formulář odeslat.

Dalším prvkem je select označený popisem "Úroveň", který nabízí tři možnosti ("první", "druhá", "třetí"). Zde se určuje o jaký se jedná stupeň v hierarchii menu.

Tento výběr je spjatý s dalšími dvěma selecty pojmenovanými jako "Pod menu", které určují pod jaké položky v menu bude menu umístěno. V případě zvolení menu první úrovně tyto selecty nemají význam, protože položka nebude pod žádné menu zařazena, ale v případě zvolení druhé nebo třetí úrovně již máme možnost kategorizace pod možné úrovně. Můžeme tedy vybírat z první a druhé úrovně.

Čtvrtým a pátým formulářovým prvkem jsou prvky textarea reprezentující textový obsah položky a poznámku, která je určena pouze pro administrátora. Obsah je také povinné vyplnit.

Posledními prvky jsou dvě tlačítka button, z nichž první umožňuje odeslat data pro nové vložení nebo pro uložení změn. Druhé funguje pouze jako vymazání/vyčištění formuláře a nastavení do stavu pro přidávání nové položky.

3.1.1.3. Výpis fotografií

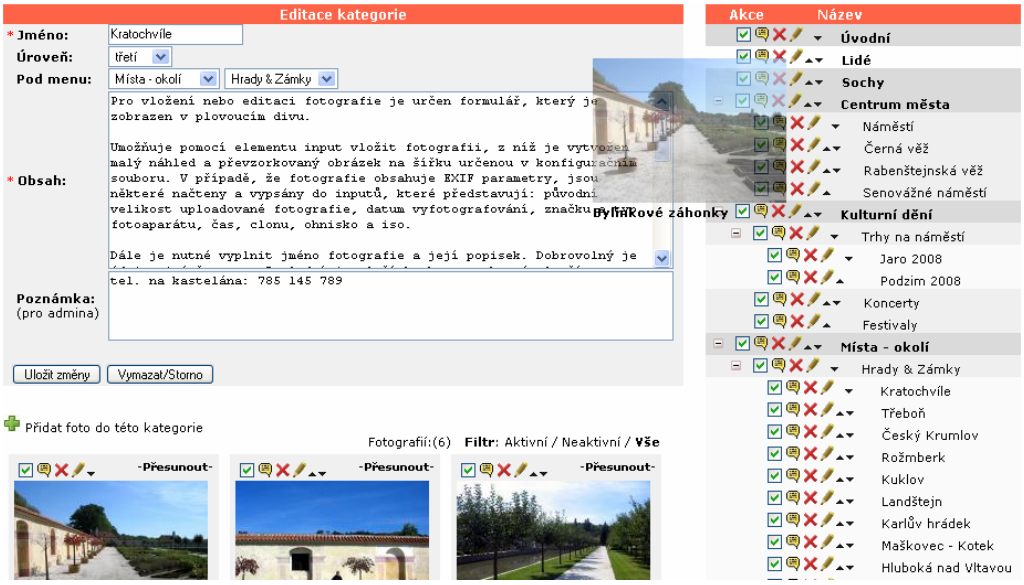
Pod formulářem pro práci s položkami menu je umístěn výpis případných fotografií (viz. Obrázek V). Tento výpis je zobrazen pouze v případě, že je formulář ve stavu pro editaci položky. To je z toho důvodu, že se nejprve musí položka vytvořit na nějaké místo v menu a poté se k ní teprve mohou přidávat fotografie.

Výpis je rozvržen po třech fotografiích na řádek. Každá fotografie je reprezentována pouze malým náhledem a jejím názvem v šedém rámečku. Dále je zde opět několik možností, které nabízejí úpravu jednotlivých fotografií. Jedná se o aktivaci, zobrazení detailu, mazání, editaci a šipky pro změnu pořadí mezi fotografiemi. Tyto funkce mají stejný význam jako v případě funkcí u položek menu.

Navíc je zde však funkce –Přesunout-, která umožňuje uchopení fotografie myší a přesunutí na nějaké místo v hlavním menu. Poté, co je nad nějakou položkou puštěno tlačítko myši, může být fotografie umístěna do této kategorie. Tímto způsobem je zajištěna neomezená možnost editace obsahu i mezi fotografiemi.

Další možností, která se objeví, je zobrazení filtru, který slouží ke zobrazování fotografií podle nastavení jejich aktivity nebo všech.

A poslední záležitost je odkaz “Přidat foto do této kategorie”, který zobrazí formulář pro práci s fotografiemi.



Obrázek VI. Náhled na přesouvání fotografie do jiné kategorie

3.1.1.4. Formulář pro práci s fotografiemi

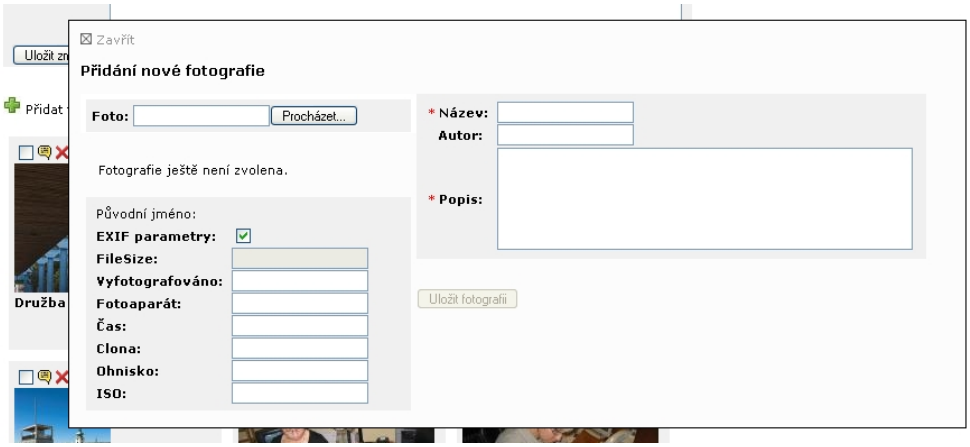
Pro vložení nebo editaci fotografie je určen formulář, který je zobrazen v plovoucím divu.

Umožňuje pomocí elementu input vložit fotografii, z níž je vytvořen malý náhled a převzorkovaný obrázek na šířku určenou v konfiguračním souboru. V případě, že fotografie obsahuje EXIF parametry, jsou některé načteny a vypsány do inputů, které představují: původní velikost uploadované fotografie, datum vyfotografování, značku a typ fotoaparátu, čas, clonu, ohnisko a iso.

Dále je nutné vyplnit jméno fotografie a její popisek. Dobrovolný je údaj o jméně autora.

Poslední je tlačítko button, které slouží pro odeslání formuláře a tím k uložení již uložené fotografie do databáze nebo k uložení změn při editaci.

Možnost zavřít a tím vynulovat hodnoty ve formuláři je odkaz zavřít nahoře v rohu.



Obrázek VII. Náhled na formulář pro práci s fotografiemi

3.1.1.4. Plovoucí divy

Posledními prvky na stránce jsou dva plovoucí divy, do kterých je zobrazován náhled na nějakou položku menu nebo detail fotografie. Pomocí funkce Drag and Drop se dají libovolně přesouvat po stránce. Slouží pouze pro informativní účely. Technicky jsou řešené podobně jako formulář pro práci s fotografiemi.

3.1.2. Soubory a adresářová struktura programu

Zde uvádím výčet a význam souborů a adresářů, které tvoří aplikaci.

Adresář/Soubor	**Popis**
./_img	Obsahuje všechy obrázky na stránce, od loga až po galerii fotografií
./content	Skripty, které reprezentují data. Ať už XML, nebo XHTML
imageFormIframe.php	Soubor potřebný pro asynchronní upload obrázku
main.page.php	Statický XHTML kód struktury stránky
xml.menu.php	PHP skript generující XML s údaji o menu

xml.photo.php	PHP skript generující XML s údaji o fotografiích
xml.selects.php	PHP skript generující XML s údaji o selectech ve formuláři
./lib	Skripty vyjadřující funkčnost webu. JavaScript, PHP, AJAXová logika
actions.inc.php	Akce, které upravují MySQL databázi
db.php	Definuje přístup k databázi
dragdrop.js	Funkce pro efekt Drag & Drop
function.ajax.js	AJAXová logika
function.inc.js	Ostatní JavaScriptové funkce
function.inc.php	Několik funkcí v PHP
login.php	Přihlášení uživatele při příchodu na web
./	
config.inc.php	Konfigurační hodnoty programu
index.php	Hlavní soubor/kostra programu
style.css	Kaskádové styly pro formát stránky

Tabulka VI. Výpis adresářů a souborů aplikace

3.2. Použité technologie

Pro realizaci aplikace jsem se rozhodl pro technologie, s nimiž jsem již měl zkušenosti.

- **PHP5**, hlavně proto, že funguje na serveru Apache, který je nadplatformní a snadno dostupný na většině poskytovaných hostinzích. Dále je v PHP5 dobrá podpora práce s XML dokumentem, databází atd…
- **MySQL** databázi. Důvod je jasný, PHP je s ní často kombinováno a toto spojení je obvyklé.
- **XHTML**
- **XML** pro přenos dat mezi serverem a prohlížečem

3.3. Představení některých funkcí

Zde se pokusím popsat zásadní funkce mé aplikace. Vzhledem k většímu rozsahu některých kódů se budu snažit vyjmout a vysvětlit vždy to podstatné a kompletní kód umístím jako přílohu této práce.

3.3.1. Vkládání položek menu z formuláře

3.3.1.1. Struktura formuláře

Pro uložení dat do databáze je určen formulář, který je naformátovaný pomocí tabulky. Jeho hlavní prvky tvoří následující elementy v tomto pořadí:

```
<input type="text" value="" name="name" id="name" />
<select name="level" id="level" onchange="setLevelMenu();">
   <option value="1">první</option>
   <option value="2">druhá</option>
   <option value="3">třetí</option>
</select>
<select name="menu1" id="menu1" disabled="disabled">
   <option value="---">---</option>
</select>
<select name="menu2" id="menu2" disabled="disabled">
   <option value="---">---</option>
</select>
<textarea name="content" id="content"></textarea>
<textarea name="desc" id="desc"></textarea>

<input type="button" id="mainButton" name="addmenu"
value="Uložit novou položku" onclick="addMenu();" />
<input type="hidden" id="idMenuLoad" name="idMenuLoad"
value="" />
```

Myslím, že vše je jasné z minulé kapitoly, jen poslední element input, který je skrytý, slouží k identifikaci akce. To jest, jestli se jedná o vkládání či editaci záznamu.

3.3.1.2. Funkce addMenu

O zavolání požadavku o uložení dat se stará funkce `addMenu`.

```
function addMenu(){
 if (xmlHttp.readyState == 4 || xmlHttp.readyState == 0){
   try{
     var name = document.getElementById('name').value;
     var level = document.getElementById('level').value;
     var menu1 = document.getElementById('menu1').value;
     var menu2 = document.getElementById('menu2').value;
     var content = document.getElementById('content').value;
     var desc = document.getElementById('desc').value;

     var postString = "addMenu=true&name=" + name +
                      "&level=" + level + "&menu1=" + menu1
                      + "&menu2=" + menu2 + "&content=" +
                      content + "&desc=" + desc;
     var url = "./content/xml.menu.php";

      xmlHttp.open("POST", url, true);
      xmlHttp.setRequestHeader("Content-
      Type","application/x-www-form-urlencoded");
      xmlHttp.onreadystatechange = function()
                        {handleRequestStateChangeMenu();};
      xmlHttp.send(postString);
     }
     catch(e){}
   }
}
```

Nejprve jsou data získáná z formuláře do proměnných (pomocí `document.getElementById`). Následně je vytvořen řetězec, ale protože se nyní pro odeslání používá metody POST, jsou vytvořeny řetězce s názvem skriptu a proměnnými zvlášť (`url` a `postString`).

Dále je nastavena funkce `handleRequestStateChangeMenu` pro událost `onreadystatechange`, která aktualizuje pravé menu. Metodou `open` nastavíme metodu POST. Rozdíl proti tomu co bylo doposud je v tom, že funkci `send` předávám místu `null` parametr `postString`.

Kód s dotazem na databázi je dostupný prostřednicvím souboru xml.menu.php. Je-li vyhodnocena podmínka `addMenu == true`. To jsme nastavili právě v řetězci `postString`.

Na stejném principu funguje rovněž ukládání dat týkajících se jednotlivých fotografií, jen s rozdílem jiného formuláře a jiných funkcí.

3.3.2. Zobrazení struktury menu

Tato funkce je velmi často volána, protože manipulace se strukturou menu je hlavní podstatou programu. Vyvolává ji tedy větší množství různých událostí. První z nich je funkce volaná událostí `onload` v elementu body při načtení stránky. Tím je zaručeno první zobrazení menu ještě před tím, než uživatel provede nějakou akci. Toto je běžný způsob.

```
<body onload="Menu();">
```

3.3.2.1. XML návrh pro přenos ze serveru

XML pro reprezentaci struktury menu tvoří soubor xml.menu.php, v němž je z databáze načtena a pomocí PHP cyklů vypsána následující struktura.

```
<menus>
   <menu>
      <title>Místa</title>
      <id>1</id>
      <id_sort>2</id_sort>
      <active>1</active>
      <description>-</description>
      <level>1</level>
      <vieworhide>-</vieworhide>
      <maxLevel>1</maxLevel>
      <submenu1>
          .
          .
          <submenu2>
               .
               .
          </submenu2>
      </submenu1>
   </menu>
</menus>
```

Každý element menu může mít libovolný počet podmenu (položek druhé úrovně), tedy elementů submenu1, které zase mohou mít libovolný počet elementů submenu2 (položek třetí úrovně). Společné jsou jim údaje title (název), id (identifikační jedinečné číslo), id_sort (pořadí ve své kategorii), active (aktivita), description (popis), level (číslo úrovně), vieworhide (ukazatel na schované podpoložky) a maxlevel (udává maximální level pro případny přesun mezi levely, aby nedošlo ke ztrátě podmenu). XML je víceméně odpovídající navržení tabulky MySQL databázi.

3.3.2.1. Funkce handleRequestStateChangeMenu

JavaScript vypisující do divu s `id="menuPrint"` tvoří tato funkce, která

obdrží předchozí XML, získá z něj hodnoty a postupně složí řetězec výsledného kódu tabulky, která menu představuje. Hlavní logiku funkce tvoří tři do sebe vnořené cykly, které projdou strukturu a každý z nich se stará o přidání jedné ze tří úrovní menu. Tato funkce je relativně dlouhá, uvádím jen výňatky.

Poté co získám do proměnné `xmlRoot` kořenový element menus, vytvořím pole všech menu první úrovně, které pak procházím cyklem pro všechny prky pole. Z nich získávám podrobné údaje a postupně vytvářím do proměnné `html` výsledný kód, když narazím na nějaké podúrovně, procházím a zpracovávám je do té doby, než je všechny projdu. Poté se vrátím o úroveň výš.

```
html="<table><tr><th>Akce</th><th>Název</th><th></th></tr>";
menuArray = xmlRoot.getElementsByTagName("menu");
for(var i = 0;i < menuArray.length;i++){
   title = menuArray.item(i).firstChild.firstChild.data;
   id = menuArray.item(i).getElementsByTagName("id").
            item(0).firstChild.data;
   html += "<tr onmouseup=\"movePhotoToMenu(" +
            photoToMenuID + ")\">";
   html += "<td>";
   html += "<img src=\"./_img/nav/edit.gif\"
            onclick=\"editMenu("+id+");\" />";

   .
   .
```

3.3.3. Upload fotografie

Asynchronní upload souboru je pomocí AJAXu velmi těžko řešitelný. Google sice problém již prý vyřešil, ale ke způsobu jak jsem se zatím nedostal Zmíněné náznaky, se kterými jsem se setkal, se mi využít nepodařilo.

Nejčastějším řešením uploadu v AJAXové aplikaci je použití malého triku s elementem iframe. Nějakým způsobem vložíme iframe do stránky a jako src ho odkážeme na soubor, v němž se nachází formulář s jedním elementem input, PHP kódem pro upload a JavaScriptem, pomocí něhož budeme přistupovat k rodičovskému dokumentu a předávat mu data.

```
<script type="text/javascript" language="javascript">
   function uploadImage(){

       var par = window.parent.document;
       var indicator = new Image(16,16);
       indicator.src = '../_img/nav/loading.gif';
       par.getElementById('loading').src = indicator.src;
       document.photoForm.submit();
    }
</script>

<form method="post" enctype="multipart/form-data">

   <strong>Foto:</strong>
   <input id="file" type="file" name="image"
                        onchange="uploadImage();" />

</form>
```

Kód funguje následovně. Přes okno, které otevře input, vyberu nějaký soubor z disku. Díky události `onchange` se spustí funkce `uploadImage`, která vytvoří přístup k rodičovskému dokumentu, ve kterém je iframe umístěn. Do něj vytvoří animovaný ukazatel, který oznamuje, že se právě nahrává soubor a umístí ho do divu s `id="loading"`. Poté odešle formulář a je zpracován PHP skript, který by měl být přístupný na začátku tohoto souboru s formulářem. Předpokládejme existující PHP funkci s názvem `uploadImage`, která vrací true/false, podle toho, jestli je úspěšně vykonána. Skript může libovolně soubor

uložit, v mém případě jen, když se jedná o JPEG. Uloží a vytvoří z něj malý náhled a jeden větší obrázek. Poté ho odstraní.

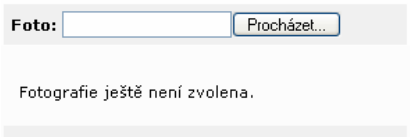
Nad formulář umístíme tento PHP skript:

```
<?
if (isset($_FILES['image'])) {
     if(uploadImage()){
          $name = "../_img/".$_FILES['image']['name'].".jpg";
          echo "<script type=\"text/javascript\">

          var par = window.parent.document;
          var imageNew = new Image();
          imageNew.src = '".$name."';
          par.getElementById('newImageThumb').src =
                                                  imageNew.src;

          </script>";
          }
    else{
          echo "<script type=\"text/javascript\">

          var par = window.parent.document;
          par.getElementById('newImageThumb').src = 'NOT';
          par.getElementById('newImageThumb').alt = 'Tento
                    soubor není možno vložit jako fotografii.';

          </script>";
    }
}
?>
```

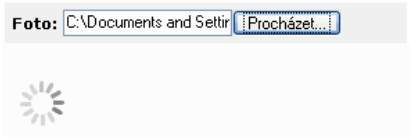
Poté co se odešle formulář je testováno, zda je uploadovaný nějaký soubor (pomocí vzniklé proměnné `$_FILES['image']` představující soubor). Dále je zkoumáno, co vrátí funkce `uploadImage`. V případě, že je soubor zpracován,

vytiskneme pomocí PHP funkce echo do stránky JavaScript, který se postará o vytvoření obrázku s náhledem a zobrazení do rodiče místo souboru loading.gif. V opačném případě se vytiskne podobný skript, bude ovšem místo vytvořeného náhledu signalizovat, že upload nebyl proveden a že se nejspíš jedná o nepodporovaný formát.

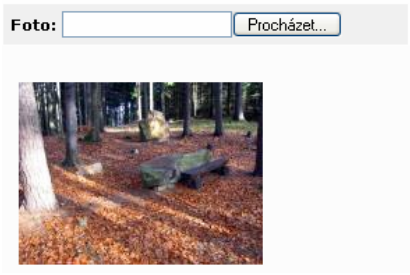
Při průběhu tohoto uploadu je tedy možné normálně na stránce pracovat a např. si vyplňovat některé údaje týkající se souboru. Poté, co bude upload dokončen, pouze se zobrazí JavaScriptem výsledek.



Obrázek VII. Před uploadem



Obrázek IX. Při uploadu



Obrázek X. Po uploadu

3.3.4. Přesun fotografie funkcí Drag and Drop

Přesun prvků na stránce pomocí myši je u RIA aplikací vítaná možnost. Navíc se ve spojení s AJAXem nabízí cesta, jak efektněji pracovat s daty a upravovat jejich ukládané vlastnosti.

V mém případě jsem potřeboval vyřešit, jak budu manipulovat s již uloženými fotografiemi v případě, že se administrátor rozhodne změnit jejich

příslušnost do kategorie. Tento požadavek jsem vyřešil právě pomocí funkce Drag and Drop (viz. Obrázek VI.).

Za kurzor myši jsem umístil prázdný div s `id="movePhoto"` s CSS vlastností `visibility: hidden;`, takže div není vidět. Celá akce se spustí prostřednictvím jakéhokoli náhledu fotografie ve výpisu galerie.

```
<span onmousedown="setPhotoToMove(9);">
   -Přesunout-
</span>
<div id="photoDrag9">
   <img alt="" src="./_img/galleryimage9.jpg"/>
   <br />
   <strong>Název obrázku č. 9</strong>
</div>
```

Poté, co vyvolá uživatel stisknutím tlačítka nad textem "-Přesunout-" událost `onmousedown`, je do divu za myší zobrazen náhled a název zvolené fotografie. K tomu slouží funkce `setPhotoToMove`.

```
var photoToMenuID;
function setPhotoToMove(photoId){

   photoToMenuID = photoId;
   document.getElementById('movePhoto').innerHTML =
   document.getElementById("photoDrag"+photoId).innerHTML;
   .
   .
}
```

Parametr funkce udává unikátní číslo id v databázi pro každou fotografii. Proto je uloženo do globální proměnné `photoToMenuID`, která udává, jakou fotografii právě přesouvám. Obsah divu za myší je tvořen duplikací položky v menu fotografií pomocí vlastnosti `innerHTML`.

Dalším krokem je identifikace id položky v menu, do kterého chci fotografii přesunout. To je zajištěno při najetí myši nad položku pomocí události `onmouseover`, kterou má definovanou každý řádek v menu. Toto id je nastaveno do globální proměnné `photoToMenu`.

```
<tr onmouseup="movePhotoToMenu()"
                onmouseover="photoToMenu = '126';">
   <td> ... </td>
</tr>
```

A konečně, jestliže pustíme nad nějakým řádkem tlačítko myši, spustí se díky události `onmouseup` funkce `movePhotoToMenu`, která volá AJAXový požadavek na server.

```
function movePhotoToMenu(idPhoto){

   if(photoToMenuID != "" && photoToMenu != ""){
       if (xmlHttp){
           try{
               var url = "./content/xml.menu.php?
                                   photoToMenu="+photoToMenu+
                                   "&idPhoto="+photoToMenuID;
               xmlHttp.open("GET", url, true);
               xmlHttp.onreadystatechange = function(){
                           handleRequestStateChangePhoto();
                           };
               xmlHttp.send(null);
            }
            catch(e){}
       }
   }
}
```

Souboru xml.menu.php jsou poslány proměnné s informacemi o tom, jakou

fotografii chci přiřadit do jakého menu. Provede se zde jednoduchý SQL dotaz, který změní ukazatel fotografie na hodnotu ukazatele na menu. Protože se provedla změna struktury menu s fotografiemi, je ještě třeba vypsat aktuální data (již bez přesunuté fotografie). Na to poslouží zpracování odpovědi funkcí `handleRequestStateChangePhoto`, která vypíše do divu pro fotografie příchozí XML data s fotografiemi, který vytvoří PHP logika v souboru (stejný způsob jako při výpisu menu kategorií).

Pro to, aby se fotografie vložila pouze v případě chtěného přesunu, musí být splněna podmínka, kdy jsou nastavené JavaScriptové proměnné `photoToMenuID` a `photoToMenu`. Takže, když pustíme tlačítko někde jinde na stránce, měla být alespoň jedna z těchto proměnných prázdná. To je zajištěno pomocí funkce `setPhotoToDefault`, která nastavuje hodnoty do stavu před přesouváním fotografie. Tedy vynuluje proměnné, vyprázdní div za myší a skryje ho. Volání této funkce jsem nastavil celému dokumentu pomocí události `onmouseup`. Tato funkce je volána i po úspěšném přesunu.

```
document.documentElement.onmouseup = setPhotoToDefault;
```

Celé přesunutí fotografie jsem představil zjednodušenou formou, pouze pro představu. Ve skutečnosti se musí ještě vyřešit to, aby se neoznačoval obsah na stránce při přesouvání fotografie. Tlačítko myši je stisknuté a pohybem se klasicky označuje vše, přes co přejedeme. I když to na funkčnost nemá vliv, nevypadá to moc dobře. Proto ve zdrojových kódech ve funkci `setPhotoToMove` jsou i kódy řešící tuto věc. Rovněž zde neuvádím kód pro správné nastavení divu vždy za kurzor.

3.3.5. Zobrazení informace o probíhající akci

U klasických desktopových aplikací platí následující pravidlo. Jestliže uživatel čeká na výsledek programu a nemá žádnou informaci, kde by viděl, že program právě vykonává nějakou akci, brzy ztrácí trpělivost. Tato informace často nemusí být nějak zvláště vypovídající a zobrazující průběh či nějaké mezistavy výpočtů. Stačí například pouze pohyblivý obrázek či podobné grafické vyjádření, které symbolizuje práci systému.

Tím, že se AJAXová stránka blíží desktopové předloze, může také vzniknout nějaké takové hluché místo, kdy se čeká na odpověď serveru, ale uživatel nevidí, že by se něco dělo. Je to způsobené tím, že na klasické stránce je odkazem stránka znovu načítána a uživatel si je jistý nějakou vyvolanou akcí a načítáním stránky. Proto se u některých AJAXových funkcí více než hodí informovat uživatele o zpracovávání požadavku. Netýká se to jen chvíle, kdy se čeká na server, ale např. i okamžiku, kdy nějaká akce proběhne a uživatel si toho ani nemusí všimnout. Proto je zde vhodné dát si práci se zprávami o dokončených požadavcích (např. "Položka byla odstraněna…").

Já jsem se rozhodl vytvořit plovoucí div, ve kterém zobrazuji animaci a nápis "loading", což mi symbolizuje zpracovávající se požadavek.

Ve své podstatě to není nic těžkého. Vyřešil jsem to tak, že jsem si dopředu připravil div s obsahem a nastavil mu CSS vlastnost `visibility: hidden;`. Pro zobrazení našeho divu je vhodná chvíle, kdy jsme odeslali požadavek, například takto:

```
xmlHttp.open("GET", url, true);
xmlHttp.onreadystatechange = function(){
                                    handleRequestState();
                                };
xmlHttp.send(null);
var loading = document.getElementById("loading");
loading.style.visibility = "visible";
```

Stačilo pouze pomocí JavaScriptu nastavit divu s `id="loading"` vlastnost `visibility` na hodnotu `visible`. Nyní bude vidět načítání dokud neuznáme za vhodné ho odstranit. To přijde v okamžiku, že akce skončila, např. jsme pomocí `innerHTML` zobrazili nějaký výsledek. Pak stačí vrátit vlastnost `visibility` do stavu `hidden` a načítání zmizí.

Obrázek XI. Loading div, při čekání na zpracování

3.4. Používané nástroje pro práci s AJAXem

Existuje řada různých nástrojů, které umožní efektivnější práci s AJAXem. Jde o nástroje pro tvorbu dokumetace, validátory, ladící prostředky nebo rozsáhlé aplikační rámce. Některé nástroje jsem vyzkoušel, některé využívám.

3.4.1. Firebug

Firebug je rozšíření pro webový browser Mozilla Firefox. Je to celkem nenápadný nástroj, který je ale schopný nabízet programátorovi opravdu velký komfort při vytváření a ladění programu. Já osobně jsem si Firebug doslova oblíbil a celou dobu, co jsem se AJAXem zabýval, jsem ho využíval dá se říct na 100%. Dnes si ani nedovedu představit ladění programu AJAXu bez Firebugu, proto mi přijde užitečné se o něm v této kapitole zmínit.

Instalace je jednoduchá, stačí stáhnout instalační soubor, který se automaticky nainstaluje do webového prohlížeče.

Firebug se objeví dole vpravo na stavové liště prohlížeče jako malá ikona. Pomocí ní můžeme zobrazit na spodní části Firefoxu okno Firebugu, díky

němuž můžeme provádět tyto zásadní funkce.

Možnost Inspect umožňuje procházet a zobrazovat detailní podrobnosti prvků ve zdrojovém kódu. Z pohledu programátora v AJAXu je příjemná možnost prohlížet si celý kód, včetně toho, který je generovaný JavaScriptem. Ten klasickým zobrazením zdrojového kódu stránky neuvidíme. Zvolenou část je možné v rámci prohlížeče libovolně editovat, ale co je pro mě podstatnější, lze nahlížet na strukturu a vlastnosti DOM každého prvku a udělat si představu o tom, jaké vlastnosti se nabízejí k použití a jaké mají atributy stav.

Obrázek XII. Firebug - procházení HTML DOM

Naprosto nejpodstatnější pro AJAXové programování je režim Console, který je schopný zobrazovat požadavky na server, HTTP hlavičky, příchozí data ze serveru a časy trvání vyřízení požadavku.

Obrázek XIII. Firebug - zobrazování AJAX požadavku a příchozích XML dat

Třetí možnost, kterou zmíním, jsou režimy Profile a Net, ve kterých je možné detailně sledovat časy a datové přenosy jednotlivých skriptů, funkcí a souborů vašeho programu.

3.4.2. Komprese Javascritového kódu

Protože je kód AJAXu stažen ze serveru v textové podobě, může si ho kdokoli zobrazit. Tím jsme v situaci, kdy každý vidí logiku, která řídí program. To má z hlediska bezpečnosti jasnou nevýhodu. Navíc použitím AJAXu nám počet JavaScriptového kódu značně vzrůstá a velikost souboru s kódem se může stát také problémem.

Existují však nástroje, které jsou schopny tyto problémy nějakým způsobem řešit. Málokdo asi ví, že pro účel nečitelnosti kódu existuje jazyk JavaScript.Encode, který je schopen tzv. obfuskace kódu. Kód je převeden do nečitelné podoby připomínající spíše byte kód, než JavaScript. V praxi se toto řešení však nepoužívá, protože má velmi malou podporu a dá se použít pouze u Internet Exploreru.

Mnohem lepší volbou je však program Memtronic, který je nabízen online na adrese *http://hometown.aol.de/_ht_a/memtronic*. Zdrojový kód je také převeden nějakým algoritmem do zcela nečitelné podoby, navíc je ale opatřen několika řádky JavaScriptu, které tento text umožní používat všem webovým prohlížečům. Dále nabízí několik úrovní komprese, kdy je náš kód zmenšen podle mých zkušeností o 50-65% (při rozsáhlejším kódu).

3.5. Problémy při práci

Jako hlavní problém, se kterým jsem se často při práci setkával, byla, jak jsem čekal, občasná rozdílná funkčnost kódu v různých prohlížečích. Snažil jsem se program testovat na běžných druzích prohlížečů a verzích. Tedy v Mozzile, Firefoxu, Opeře, Netscape Navigatoru a Internet Exploreru. I když jsem musel několikrát kvůli některému z nich kód měnit, nakonec se mi podařilo vyrobit fungující verzi programu. Programátor se jen musí smířit s tím, že některé JavaScripty bude muset opravdu napsat v několika verzích pro

různé prohlížeče. Takovéto kódy rostou úměrně se tvorbou funkcí jako Drag and Drop, ale i některé údaje, k nimž přistupujeme pomocí DOM JavaScriptu, občas vykazují divné chování.

Další nevýhodou je to, že při chybě skriptu na serveru není v případě použití XML vidět přímo chybová hláška nikde ve stránce. To je však možné ošetřit. Buď si vytvořit nějakou funkci pro zobrazování chyb nebo se spolehnout např. na zobrazení ve zmiňovaném Firebugu.

Jako jistý problém se ukázal být upload souboru na server. Toto je pomocí AJAXu komplikované. Existuje několik různých polofunkčních řešení, se kterými jsem se setkal. Problém je vždy hlavně v nefunkčnosti ve všech prohlížečích. Pro tento účel je nejčastěji používán malý trik pomocí elementu iframe. Dá se říct, že dobrat se k řešení tohoto problému znamenalo u mě z časového hlediska větší problém.

Sehnat vhodný server, na který bych program umístil, bylo dalším problémem. Je třeba si dát pozor na to, jestli je aktivní PHP rozšíření pro práci s XML. Způsob zpracování XML pomocí PHP prodělal velké změny a jestliže využíváme PHP DOM funkce, je třeba si zjistit jejich podporu a vyhledat novou verzi PHP.

Protože pracuji s obrázky i větších rozlišení, byly trochu potíže se sehnáním serveru, který má dostačující paměťový limit pro mé skripty. Celý proces nahrání a převzorkování fotografie (příp. vložení vodoznaku) vyžaduje nastavenou direktivu serveru Apache memory_limit na hodnotu alespoň 20Mb.

4. Závěr

Myslím, že se mi úspěšně podařilo splnit cíle, které jsem si předeslal. Popsal jsem rozšiřující se techniku AJAX a věci, které s ní nějak souvisí. Celou dobu jsem se snažil zmiňovat hlavně záležitosti, se kterými jsem měl já osobně při učení potíže a popisovat způsoby, kterými jsem se rozhodl řešit situace, ve kterých je AJAX nasazován. Vždy jsem se snažil odůvodnit, proč jsem se pro konkrétní řešení rozhodl a i v kapitolách, které na první pohled s AJAXem přímo nesouvisí, vždy vysvětluji jejich důležitost a důvod k jejich umístění do tohoto textu.

Vytvořil jsem praktickou aplikaci, která je funkční a je na ní vidět, jak je možné pomocí AJAXu vylepšit zaběhlé postupy. Podařilo se mi vytvořit rozhraní, jehož nároky jsem si předem stanovil, a které je nyní připravené uplatnit se v praxi.

S tím, jak jsem se s technikou AJAX postupně seznamoval, musím konstatovat, že plně souhlasím se všemi zmiňovanými výhodami nasazení a přikládám se také k názoru, že AJAX čeká velká budoucnost.

I přes časté komplikace, které mě ze začátku mou prací provázely, myslím, že i já se v budoucnu přidám k programátorům, kteří se rozhodli pro AJAXový výhodný přístup k některým problémům, které se na běžném webu vyskytují.

Přílohy

[1] Zdrojové kódy programu na přiloženém CD

[2] Výčet často používaných událostí JavaScriptu

Název události	Popis/ kdy nastává
onClick	Při kliknutí myši
onDblClick	Při dvojkliku myši
onMouseOver	Při najetí kurzoru myši nad element
onMouseOut	Při opuštění elementu kurzorem myši
onMouseDown	Při stisknutí tlačítka myši
onMouseUp	Při uvolnění tlačítka myši
onMouseMove	Při pohybu kurzoru myši

Události pro práci s myší

Název události	Popis/ kdy nastává
onAbort	Při nedokončení načítání objektu
onError	Když nemůže být objekt načten
onLoad	Po nahrání celého obsahu elementu
onUnLoad	Po odebrání obsahu z okna
onResize	Při změně velikosti obrazovky
onScroll	Při scrollování obrazovky

Události pro práci s HTML dokumentem

Název události	Popis/ kdy nastává
onSubmit	Při odeslání formuláře
onReset	Při smazání obsahu tlačítkem reset
onFocus	Při aktivaci prvku formuláře
onBlur	Při deaktivaci prvku formuláře
onChange	Při změně obsahu prvku formuláře
onSelect	Při výberu textu v prvku formuláře

Události pro práci s formuláři

Název události	Popis/ kdy nastává
onKeyDown	Při stisku klávesy
onKeyUp	Po uvolnění klávesy
onKeyPress	Při stisku a uvolnění klávesy

Události pro práci s klávesnicí

Použité zdroje

[1] ASLESON, Ryan, SCHUTTA, Nathaniel T. *AJAX - vytváříme vysoce interaktivní webové aplikace*. [s.l.] : Computer Press, 2006. 269 s.

[2] DARIE, Cristian, et al. *AJAX a PHP - tvoříme interaktivní webové aplikace profesionálně*. [s.l.] : Zoner Press, 2006. 320 s.

[3] PICHLÍK, Roman. *Interval.cz* [online]. 2005 [cit. 2008-01-04]. Dostupný z WWW: <http://interval.cz/clanky/rich-internet-application>.

[4] VRÁNA, Jakub. *Php.vrana.cz* [online]. 2005 [cit. 2008-01-04]. Dostupný z WWW: <http://php.vrana.cz/ajax.php>.

[5] KOSEK, Jiří. *Kosek.cz* [online]. 1999 [cit. 2008-02-17]. Dostupný z WWW: <http://www.kosek.cz/clanky/xml/ >.

[6] *Webtvorba.cz* [online]. 2004 [cit. 2008-02-17]. Dostupný z WWW: <http://www.webtvorba.cz/css/>.

[7] *Webtvorba.cz* [online]. 2004 [cit. 2008-02-17]. Dostupný z WWW: <http://www.webtvorba.cz/xhtml>.

[8] *Json.org* [online]. [cit. 2008-04-04]. Dostupný z WWW: <www.json.org/json-cz.html>.