Pedagogická fakulta
Faculty of Education

Jihočeská univerzita v Českých Budějovicích
University of South Bohemia in České Budějovice

Jihočeská univerzita v Českých Budějovicích
Pedagogická fakulta
Katedra informatiky

Bakalářská práce

Drag and Drop API v HTML5

Vypracoval: Vít Barabáš, DiS.
Vedoucí práce: PaedDr. Petr Pexa, Ph.D.

České Budějovice 2013

Anotace

Práce se zabývá novým způsobem ovládání webových aplikací pomocí techniky "Chyť a pusť" v jazyku HTML5.

Práce se skládá ze dvou částí. První část práce se věnuje popisu DND API v HTML5. Součástí této části je analýza podpory napříč běžně používanými prohlížeči. Na první část navazuje praktická část skládající se z návrhu a následné realizace fotogalerie využívající DND API.

Klíčová slova:

Chyť a pusť, HTML5, AJAX, fotogalerie, jQuery

Abstract

The work (the bachelor´s thesis) deals with a new way of web application management via the "drag and drop" technique in the HTML5 programming language.

The work is divided into two parts. The first part consists of DND API description in HTML5. The support analysis within common web browsers is included as a part of this description. The second, practical part of the thesis focuses on a concept and the following realisation of a photogallery using DND API.

Keywords:

Drag and Drop, HTML5, AJAX, photogalery, jQuery

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, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích

dne: ....

Podpis autora práce

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH
Fakulta pedagogická
Akademický rok: **2011/2012**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Vít BARABÁŠ**
Osobní číslo:	**P10386**
Studijní program:	**B7507 Specializace v pedagogice**
Studijní obor:	**Informační technologie ve vzdělávání**
Název tématu:	**Drag and Drop API v HTML5**
Zadávající katedra:	**Katedra informatiky**

Zásady pro vypracování:

Cílem bakalářské práce bude zpracování dostupných informací o nové a perspektivní technologii Drag and Drop API v HTML5 a provedena analýza možného použití ve webové prezentaci. Bude otestována podpora napříč běžně používanými prohlížeči a proveden průzkum situace na trhu s on-line fotogaleriemi (porovnání např. s Google Picasa, rajce.net, jAlbum apod.). Na základě této analýzy bude provedeno dotazníkové šetření mezi běžnými uživateli i administrátory webových prezentací. Praktickou částí práce bude vytvoření plně dynamické a dotykově ovládané webové fotogalerie postavené na technologiích PHP/MySQL/Drag and Drop API/HTML5 s důrazem na přívětivost a intuitivnost administračního rozhraní. Veškeré úpravy (tvorba náhledů, změna velikosti, rozlišení apod.) budou automaticky provedeny na straně klienta bez zatěžování serveru.

Rozsah grafických prací:	**CD ROM**
Rozsah pracovní zprávy:	**60**
Forma zpracování bakalářské práce:	**tištěná**
Seznam odborné literatury:	**viz příloha**

Vedoucí bakalářské práce: **PaedDr. Petr Pexa, Ph.D.**
Katedra informatiky

Datum zadání bakalářské práce: **12. dubna 2012**
Termín odevzdání bakalářské práce: **26. dubna 2013**

Mgr. Michal Vančura, Ph.D.
děkan

L.S.

doc. PaedDr. Jiří Vaníček, Ph.D.
vedoucí katedry

V Českých Budějovicích dne 12. dubna 2012

Příloha zadání bakalářské práce

Seznam odborné literatury:

1. W3.ORG. Drag and Drop API [online]. 25.5.2011 [cit. 2012-03-25]. Dostupné z: http://www.w3.org/TR/html5/dnd.html#dnd
2. W3SCHOOLS.COM. Drag and Drop [online]. 25.3.2012 [cit. 2012-03-25]. Dostupné z: http://www.w3schools.com/html5/html5_draganddrop.asp
3. HTML5DEMOS.COM. Drag and Drop Demos: automatic upload [online]. 25.3.2012 [cit. 2012-03-25]. Dostupné z: http://html5demos.com/dnd-upload
4. JOUANNEAU, Laurent. Drag and drop demo in a HTML document: using the HTML5 drag and drop API [online]. 25.3.2012 [cit. 2012-03-25]. Dostupné z: http://ljouanneau.com/lab/html5/demodragdrop.html
5. ZACHAŘ, Jiří. JavaScript: plynulá změna velikosti obrázku [online]. 15.3.2008 [cit. 2012-03-25]. Dostupné z: http://www.zaachi.com/cs/items/javascript-plynula-zmena-velikosti-obrazku.html
6. MOZILLA.ORG. How to develop a HTML5 Image Uploader [online]. 5.1.2011 [cit. 2012-03-25]. Dostupné z: http://hacks.mozilla.org/2011/01/how-to-develop-a-html5-image-uploader/
7. STACKOVERFLOW.COM. Resizing an image in an HTML5 canvas [online]. 23.2.2010 [cit. 2012-03-25]. Dostupné z: http://stackoverflow.com/questions/2303690/resizing-an-image-in-an-html5-canvas
8. ORGOŇ, Vojtěch. ZDROJAK.CZ. Upload obrázků pomocí HTML5 [online]. 2.2.2011 [cit. 2012-03-25]. Dostupné z: http://zdrojak.root.cz/clanky/upload-obrazku-pomoci-html5/

Poděkování

Rád bych touto formou poděkoval vedoucímu mé závěrečné práce PaedDr. Petru Pexovi, Ph.D. za odborné vedení při zpracování práce a za cenné rady a doporučení, bez kterých bych tuto práci nenapsal.

Dále bych chtěl poděkovat mému otci, který mě v průběhu psaní této práce podporoval, a tím mi umožnil práci dokončit.

Obsah

1 Úvod

Webové aplikace se těší čím dál větší oblibě, a to jak mezi uživateli podnikové sféry, tak i mezi běžnými koncovými uživateli internetu. Obliba představuje přirozený důsledek mnoha výhod, který tento způsob distribuce programů poskytuje. Mezi největší výhody patří nezávislost programu na platformě a snadná distribuce bez nutnosti instalovat program do počítače.

V mé bakalářské práci se zaměřím na nové trendy ovládání pokročilých a interaktivních aplikací. Dle mého názoru největší nevýhodou současných webových aplikací je zatím poněkud omezený způsob ovládání. Stejně jako se ovládání programů metodou "Chyť a pusť" ujalo při ovládání v dnešních operačních systémech, věřím, že se tento princip ujme i při ovládání webových aplikací.

1.1 Cíle práce

Bakalářská práce má za úkol zpracovat dostupné informace o moderní technologii DND API v HTML5. Dále se zabývá otestováním podpory této techniky mezi aktuálními prohlížeči na konkrétních příkladech použití DND API. Pro toto testování jsou navrženy 3 testovací úlohy, na kterých bude následně vyzkoušena podpora čistého HTML5 DND API dle specifikace W3.org.

Praktická část práce zahrnuje navržení a naprogramování fotogalerie využívající metody ovládání "Chyť a pusť". Aplikace bude umožňovat nahrání fotografie na server přetáhnutím do prohlížeče s fotogalerií. Fotografie bude následně upravena dle nastavení serveru do požadované velikosti a nahrána na server. Aplikace by měla umožnit uživateli udělat výřez a automaticky změnit rozlišení na takové, které odpovídá nastavení galerie.

Fotografie se mohou řadit přetažením jednotlivých fotografií v obou osách, přetažením celých řádků s fotografiemi nebo přetažením fotografií po sloupcích.

1.2 Metoda vývoje aplikace

Při vývoji softwaru může jeho tvůrce postupovat několika způsoby. Příkladem může být třeba tzv. ADDIE model. Tento model vývoje umožňuje efektivně postupovat při vývoji v pěti vzájemně provázaných krocích a snížit tak rizika

výskytu problému na minimum. Dle tohoto modelu budu postupovat při vývoji mé práce i já.

1.2.1 Analýza

Během této fáze dojde ke zkompletování literatury potřebné pro vytvoření fotogalerie, jako HTML5 DND API, HTML5 SELECTORS API či AJAX. Všechny zpracované informace poslouží k sestavení principu práce s nimi. Bližší popis bude zahrnovat kapitola s názvem "Terminologie a popis použitých technologií".

V dalším kroku dojde k navrhnutí testovacích příkladů využívajících DND API a další technologie HTML5. Na těchto příkladech se otestuje podpora technologií mezi běžnými prohlížeči. Výsledky testu budou zpracovány v kapitole "Test podpory DND API mezi prohlížeči".

1.2.2 Návrh

Na základě analýzy vypracuji myšlenkovou mapu, která pomůže při kontrole plnění cílů a zároveň slouží jako vodítko při vývoji aplikace. Mapu bude možno nalézt v příloze této práce.

Dle sestaveného harmonogramu provedu návrh jádra aplikace a definuji funkcionality, které má fotogalerie obsahovat. Rovněž budou navrženy datové modely potřebné pro komunikaci v rámci aplikace a drátěné modely rozhranní administrace.

Další nezbytnou částí je návrh databáze. Databáze se navrhne pomocí ER databázového modelu tak, aby splňovala požadavky, které jsou nutné pro dosažení cílů.

1.2.3 Vývoj

Během této fáze naprogramuji samotnou aplikaci. Vývoj aplikace rozdělím do několika menších částí dle metody "rozděl a panuj". v těchto etapách vyřeším dílčí úkoly a po dokončení každé etapy proběhne konzultace s vedoucím práce, za jehož rady jsem mu nesmírně vděčný.

1.2.4 Implementace

Po vypracování předchozích kroků byla práce konzultována a nabídnuta k odzkoušení ve firmě FTSun s.r.o. zabývající se vývojem webových aplikací na míru. Dále byla aplikace uveřejněna na mých internetových stránkách www.vituvsvet.cz, kde je dostupná k volnému stažení. Aplikaci použiji pro komerční účely na několika nových projektech.

1.2.5 Evaluace

Aplikace byla nabídnuta v rámci uzavřeného beta testu vybraným studentům Pedagogické fakulty Jihočeské univerzity a několika kamarádům. Zároveň jsem všechny požádal o zpětnou vazbu, jak se jim aplikace líbila. a čekal jsem na případné nalezené chyby.

Po prvních testech aplikace mi byla nahlášena celá řada chyb. Všechny nahlášené chyby jsem opravil a za tyto nahlášené problémy jsem všem velice vděčný. Bez jejich pomoci bych je opravoval mnohem delší dobu.

2 Terminologie a popis použitých technologií

Ačkoli by se mohlo zdát, že je tento princip ovládání webových stránek zcela nový, DND API v HTML5 je postavené na specifikaci DND API v Internet Explorer 5. Tento způsob ovládání implementoval Microsoft jako první. Na rozdíl od původního API, v HTML5 specifikaci programátor již nemusí používat nízko-úrovňové události typu **mousedown** a **mousemove**, ale může použít události typu **dragstart** a **dragend**.

Pokud chceme využívat výhod metody ovládání Drag and Drop API v HTML5, je třeba si toto API popsat. Zároveň je třeba pochopit samotný princip práce s ním. Tato kapitola se bude zabývat právě popisem tohoto API.

2.1 Princip DND API

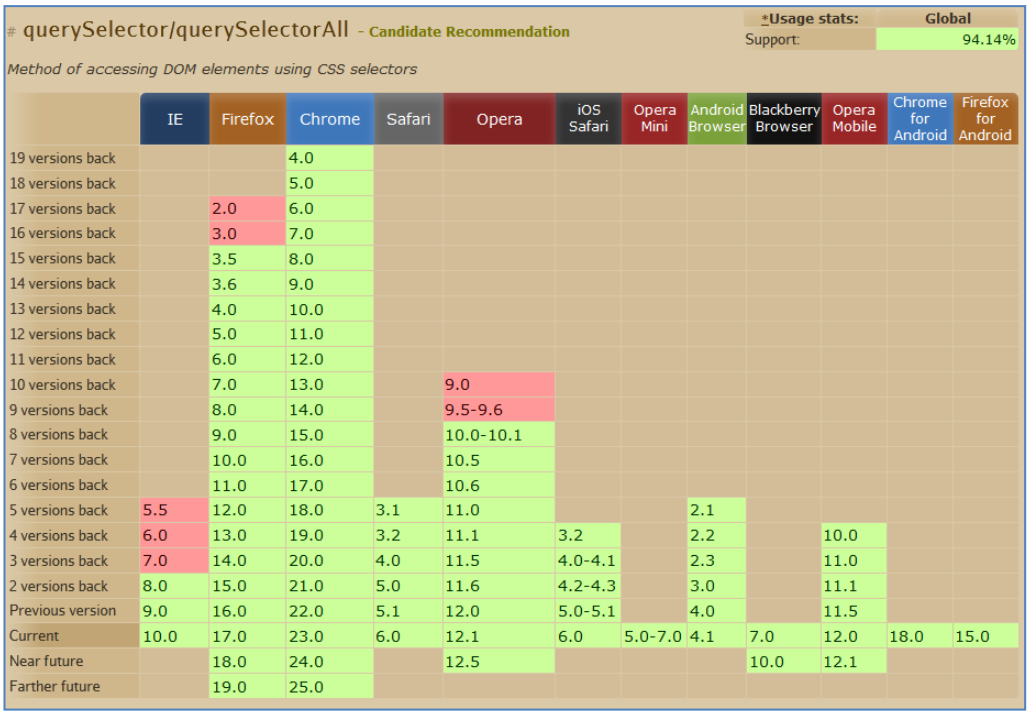
Základním principem použití DND API je označení elementu, u kterého chceme provádět táhnutí, atributem **draggable**. Dále je potřeba nastavit tomuto elementu událost **dragstart**.

2.1.1 Identifikace elementu

Události se musí registrovat na konkrétní elementy v DOM objektu, proto je nutné provést výběr elementů, které plánujeme používat pro táhnutí. První možností je použití HTML5 SELECTORS API, nebo lze použít nějaký framework jako např. jQuery. Při ukázkových příkladech, na kterých budu testovat prohlížeče, bude použito HTML5 SELECTORS API. Dle obr. 1 je podpora mezi prohlížeči poměrně dobrá. v době psaní této práce dosáhla zhruba 94%.

Vyhledání elementu je realizováno metodami **querySelector** a **querySelectorAll**. První metoda z výše uvedených vrací první odpovídající výskyt elementu včetně vnořených elementů. Druhá metoda vrací pole elementů odpovídajících hledanému řetězci.

Tento řetězec odpovídá Selector API Level 4 dle specifikace W3.org a jeho detailní popis by byl nad rámec této kapitoly. Celá specifikace je dostupná na webu W3.org ([4]). Pro potřebu této práce bude stačit práce s elementy, třídami a identifikátory. Hledání elementu patřícího do určité třídy se provede podobně jako je tomu při výběru v jazyce CSS, a to znakem tečky následovaným jménem třídy. Řetězci, kterým se provádí výběr, se říká **query string,** dále jen QS.



querySelector/querySelectorAll - Candidate Recommendation

*Usage stats: Global

Support: 94.14%

Method of accessing DOM elements using CSS selectors

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	Opera Mobile	Chrome for Android	Firefox for Android
19 versions back			4.0									
18 versions back			5.0									
17 versions back		2.0	6.0									
16 versions back		3.0	7.0									
15 versions back		3.5	8.0									
14 versions back		3.6	9.0									
13 versions back		4.0	10.0									
12 versions back		5.0	11.0									
11 versions back		6.0	12.0									
10 versions back		7.0	13.0		9.0							
9 versions back		8.0	14.0		9.5-9.6							
8 versions back		9.0	15.0		10.0-10.1							
7 versions back		10.0	16.0		10.5							
6 versions back		11.0	17.0		10.6							
5 versions back	5.5	12.0	18.0	3.1	11.0			2.1				
4 versions back	6.0	13.0	19.0	3.2	11.1	3.2		2.2		10.0		
3 versions back	7.0	14.0	20.0	4.0	11.5	4.0-4.1		2.3		11.0		
2 versions back	8.0	15.0	21.0	5.0	11.6	4.2-4.3		3.0		11.1		
Previous version	9.0	16.0	22.0	5.1	12.0	5.0-5.1		4.0		11.5		
Current	10.0	17.0	23.0	6.0	12.1	6.0	5.0-7.0	4.1	7.0	12.0	18.0	15.0
Near future		18.0	24.0		12.5				10.0	12.1		
Farther future		19.0	25.0									

OBR. 1 PODPORA HTML5 SELECTORS API MEZI PROHLÍŽEČI [1]

Pokud hledáme první výskyt elementu div, který patří do třídy "test", QS by mohl vypadat následovně:

```
var num = document.querySelector('.test');
```

Identifikace elementu probíhá těmito způsoby:

1. přiřazením identifikátoru pomocí atributu **id** (pro jednoznačnou identifikaci konkrétního elementu - QS ve formátu "#jméno_id")
2. nastavením třídy pomocí atributu **class** (pro skupinu elementů QS ve formátu ".jméno_třídy")
3. nastavením události na konkrétní typ elementu (QS ve formátu "jméno_elementu")
4. kombinací bodu 3 s ostatními (QS ve formátu "p.trida" či "div#id")

[1] Method of accessing DOM elements using CSS selectors. *Http://www.caniuse.com* [online]. [cit. 2012-12-07]. Dostupné z: http://caniuse.com/#search=querySe

2.1.2 Registrace události

Nastavení události pro konkrétní element probíhá pomocí skriptu (JavaScript). Zde ovšem začíná první problém, a tím je rozdílná implementace funkce pro nastavení události elementu v DOM objektu. Zatímco prohlížeče postavené na jádru webkit (Safari, Google Chrome) či Mozilla Firefox používají v DOM objektu pro registraci události metodu **addEventListener**, prohlížeč Internet Explorer používá metodu **attachEvent**.

Tento problém za programátora řeší knihovna h5utils.[2] v příkladech pro testování podpory DND API použiji právě tuto knihovnu. Výhodou této knihovny je minimální velikost kódu a tím i úspora systémových zdrojů klientského počítače. Další možností, jak tuto nekompatibilitu mezi prohlížeči vyřešit, je použití některého z hotových JavaScript frameworků jako je např. jQuery, který bude použit při vývoji fotogalerie. Nevýhodou těchto frameworků může být vyšší náročnost na systémové zdroje klienta, protože se kromě používaného kódu načítá celý balík funkcí daného frameworku.

Knihovna h5utils přidává metodu **addEvent**. Tato metoda umožňuje registrovat událost pro konkrétní element nezávisle na prohlížeči. Metoda vyžaduje 3 parametry, a to element, pro který se událost nastavuje, druh události a funkci, jenž se provede při vyvolání události. Samotný výběr a nastavení události by pak mohlo vypadat takto:

```
// proměnná uchovávající počet začátků táhnutí
var pocetTahnuti = 0;
// funkce, která se spustí při začátku táhnutí
function startDrag(event) {
 pocetTahnuti++;
}
var el = document.querySelector('#id_nazev');
// registrace události
addEvent(el, 'dragstart', startDrag);
```

[2] Knihovna h5utils je dostupná z adresy http://html5demos.com/js/h5utils.js

Pokud tento kód otestujeme v prohlížeči Mozilla Firefox, zjistíme, že se prohlížeč chová jinak než ostatní prohlížeče. Na první pohled se může zdát, že blok táhnout nejde. Toto chování je způsobeno výchozím nastavením dat, která se předávají při události táhnutí. Zatímco ostatní prohlížeče tyto data nastavit nevyžadují, Mozilla Firefox bez tohoto nastavení nefunguje. Upravená funkce **startDrag** bude vypadat takto:

```
function startDrag(event) {
 pocetTahnuti++;
 // data předávaná při táhnutí
 event.dataTransfer.setData('Text',
      this.id);
}
```

Takto upravený kód již funguje v prohlížeči Mozilla Firefox, ale nebude fungovat v prohlížečích Internet Explorer 9 a starších, u kterých lze tuto metodu aplikovat pouze na elementy obrázků či odkazů. Tato nefunkčnost je způsobena neúplnou implementací DND API tak, jak jej definuje W3.org, které lze použít pro všechny elementy. Tím je celé použití API v těchto prohlížečích velice omezené. Rozšířené použití na bloky a ostatní elementy podporuje Internet Explorer až od verze 10. Kompletní zdrojový kód příkladu naleznete jako přílohu 01_dragstart.

2.1.3 Reakce na přetažení

Samotné táhnutí elementy by nebylo příliš užitečné bez reakce ostatních elementů na tyto objekty. HTML5 DND API poskytuje několik událostí, na které lze reagovat. První takovou událostí je událost **dragenter**.

Událost dragenter se spouští v momentě, kdy je nad reagující objekt přetažen jiný tažitelný objekt. Pokud je taženým objektem v rámci jednoho táhnutí opuštěn kolizní prostor reagujícího objektu, a poté je táhnuto taženým objektem zpět do kolizního prostoru reagujícího objektu, dojde k opětovnému vyvolání události.

Pokud se pokusím upravit předchozí příklad tak, aby blok s id "test2" reagoval na jiné objekty, kód bude rozšířen následovně (celý kód je dostupný v příloze 02_dragenter).

```
// výběr objektu reagující na ostatní elementy
var pocetKolizi = 0;
var cil = document.querySelector('#test2');
// zkrácený zápis registráce události,
// k předchozímu zápisu je ekvivalentní
addEvent(cil, 'dragenter', function (event){
 // inkrementace počítadla kolizí
 pocetKolizi++;
 // nastavení obsahu elementu "test2"
 this.innerHTML = 'Na tento objekt byl
 přetáhnut jiný element. ' +
 pocetKolizi + 'x';
 });
```

Na tomto příkladě je vidět zkrácená forma zápisu, která je ekvivalentní k předchozímu zápisu.

Funkcí velice podobnou události je událost **dragover**, která se liší od události dragenter pouze četností volání v rámci jedné kolize mezi objekty. Zatímco dragenter je volána pouze při první detekci kolize a další až po opuštění a opětovném najetí, událost dragover je volána ve smyčce, dokud nedojde k opuštění kolizního prostoru reagujícího objektu či uvolnění taženého objektu. Tuto událost ověříme například takto (kód je dostupný v příloze 03_dragover):

```
addEvent(cil, 'dragleave', function (event){
 this.innerHTML = this.innerHTML + 'X';
});
```

Příklad se chová tak, že po vyvolání kolize mezi objekty kód začne rozšiřovat obsah bloku "test2" písmeny X.

Další událost, která umožňuje interakci mezi objekty je událost **dragleave**. Tato událost se spouští vždy v momentě, kdy tažený objekt opouští kolizní prostor reagujícího objektu. Typickým použitím této události může být uvedení reagujícího objektu do původního stavu před tažením nad reagující objekt. Registrace události

probíhá ve stejném duchu jako ty předchozí (celý kód je dostupný v příloze 04_dragleave).

```
addEvent(cil, 'dragleave', function (event){
 this.innerHTML = 'Nelze jím táhnout, ale
 reaguje na jiné objekty';
});
```

Velice důležitou událostí pro používání HTML 5 DND API je událost **drop**. Tuto událost je třeba nastavit na reagujícím objektu a je vyvolána ve chvíli, kdy dojde k uvolnění taženého objektu v kolizní zóně reagujícího objektu. Událost se hodí pro ovlivnění reagujícího či taženého objektu po dokončení táhnutí. Příklad použití události by mohl vypadat takto:

```
function startDrag(event) {
 pocetTahnuti++;
 event.dataTransfer.setData('Id', this.id);
}
addEvent(cil, 'drop', function (event){
 event.preventDefault();
 var data=event.dataTransfer.getData("Id");
 event.target.style.backgroundColor = 'blue';
 var el = document.querySelector('#' + data);
 el.style.backgroundColor = 'red';
});
```

Funkce pro registraci události drop obsahuje parametr typu DragEvent. Tento objekt obsahuje ukazatel na konkrétní reagující objekt, na který byl přetažen jiný objekt. Pomocí tohoto ukazatele lze nastavit všechny vlastnosti objektu, jak ukazuje příklad **05_drop**. Jedná se o modrou barvu pozadí.

Objekt typu DragEvent zahrnuje objekt typu DataTransfer, který umožňuje předávat informace od taženého objektu směrem k reagujícímu objektu. k tomuto přenosu informace se využívají dvě metody objektu DataTransfer. První metodou je metoda setData(index, hodnota), kterou se data nastavují. v našem případě si lze při události dragStart ukládat různé informace jako například identifikátor taženého

objektu. Druhou metodu představuje metoda getData(index), která zajišťuje čtení přenášených dat dle konkrétního indexu. Tyto informace lze nastavovat i číst z objektu při různých událostech táhnutí. Samotné nastavení vlastnosti na taženém objektu je realizováno přes selektor API standardní cestou.

2.1.4 Nastavení taženého tvaru

Aby byl výčet funkcionality DND kompletní, je třeba zmínit jednu funkcionalitu. Jedná se o nastavení taženého tvaru. Jako základní tvar při táhnutí je použit ten tvar, který má objekt před tažením. To nemusí být vždy praktické, proto je v objektu DataTransfer obsažena metoda setDragImage(IMGElement, offsetX, offsetY). Touto metodou můžeme nastavit místo taženého elementy jakýkoli obrázek. Prvním parametrem metody může být jakýkoli img element, canvas element či input s atributem type=image a další dva parametry představují posunutí v pixelech na osách X a Y od špičky kurzoru myši. Upravená funkce startDrag by mohla vypadat takto:

```
function startDrag(event) {
 var dragIcon = document.createElement('img');
 dragIcon.src = kocka.png';
 dragIcon.width = 100;
 event.dataTransfer.setDragImage(dragIcon, -10, -
10);
 event.dataTransfer.setData('Id', this.id);
}
```

Jak je vidět, změna obrázku není složitá. v budoucnu by mohla být dostupná i sada CSS pseudo tříd drag, drag-over, drag-enter apod., díky kterým by bylo možné přímo stylovat táhnutý element beze změny původního elementu. Prvním, kdo tyto pseudo třídy implementuje, jsou tvůrci prohlížeče Mozilla Firefox, ovšem zatím za pomoci prefixu "-moz-"[3]. Jiné prohlížeče je bohužel zatím neimplementují, a proto se v době psaní této práce nehodí pro nasazení.

[3] Zmínka o plánované pseudo třídě zde: :-moz-drag-over. *Developer.mozilla.org* [online]. 2011 [cit. 2013-03-03]. Dostupné z: https://developer.mozilla.org/en-US/docs/CSS/:-moz-drag-over

2.2 Asynchronní komunikace

Každá moderní webová aplikace potřebuje komunikovat se serverem, a to ideálně bez nutnosti kompletního přečtení celé webové stránky. Zatímco staré aplikace využívaly pro výměnu dat statické stránky napsané v HTML, ty modernější dokázaly za pomoci serverových skriptů jako ASP, PHP či CGI generovat obsah dynamicky. Sice se při každém dotazu musel načíst celý obsah znovu, ale opravdová interakce dorazila až s možností asynchronní komunikace mezi klientskou aplikací a serverem.

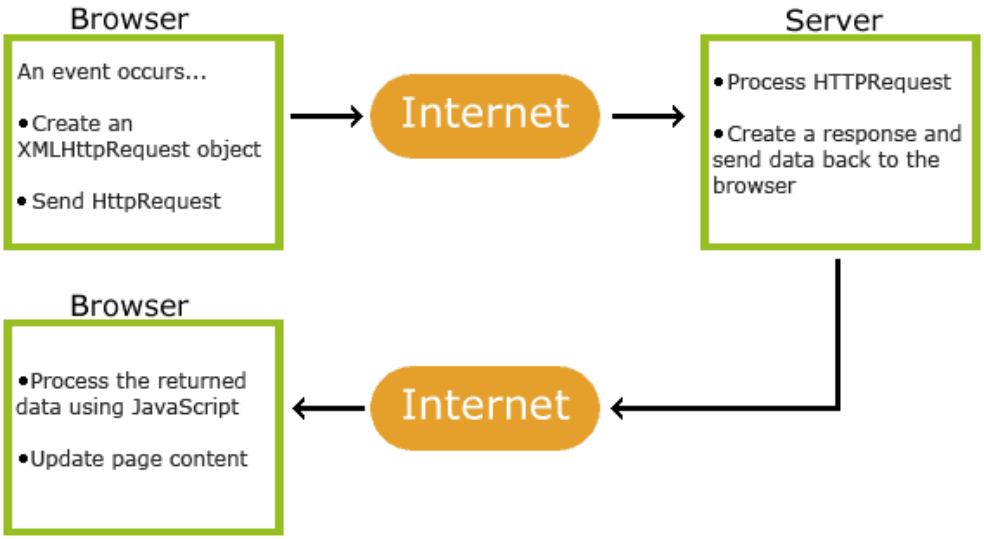
2.2.1 Web Sockets

Nejnovější technologií, kterou nám pro asynchronní komunikaci HTML5 přináší, je technologie Web Sockets. v zásadě se jedná o komunikační spojení navázané mezi dvěma procesy pomocí IP technologie. Socket je (zjednodušeně řečeno) popsán dvojicí IP adres, protokolem a portem, který je pro komunikaci použit. Služby na vyšší úrovni tento socket používají k (většinou obousměrnému) přenášení dat ([11]). Technologie na rozdíl od jiných asynchronních technologií umožňuje opravdovou online oboustrannou komunikaci a není třeba zajistit na straně klienta nekonečnou smyčku, která se dotazuje serveru, zda se něco nového nestalo.

Podpora mezi aktuálními prohlížeči se velice zlepšila. Běžné desktopové prohlížeče už tuto technologii v nejnovějších verzích podporují, avšak u mobilních prohlížečů je stále podpora velice slabá. Jako opravdový problém této technologie se podle mého názoru v současnosti jeví to, že tato technologie vyžaduje speciální vybavení na straně serveru. Vzhledem ke špatné rozšířenosti tohoto vybavení mezi běžně dostupnými hostingy je tato technologie v současnosti nepoužitelná. Pokud chcete web sockets používat na serveru Apache s PHP, musíte na serveru doinstalovat rozšiřující knihovnu **pywebsocket** napsanou v Pythonu a v konfiguraci Apache povolit rozšíření **mod_websocket**.

2.2.2 AJAX

AJAX neboli "Asynchronous JavaScript and XML" je technika pro vytvoření rychlých a dynamických webových stránek. AJAX umožňuje webové stránce aktualizovat se asynchronně při zachování malého přenosu dat. Tato technika umožňuje načítat jednotlivé části webové stránky, aniž by se musela načíst celá stránka ([12]). Princip přenosu dat je popsán na Obr. 2.



Obr. 2 Princip přenosu dat pomocí techniky AJAX[4]

Podle Obr. 2 tato technologie využívá objektu XMLHttpRequest, který je schopen navázat spojení se vzdáleným serverem a po vrácení odpovědi vyvolat události v prohlížeči, a tak změnit obsah webové stránky. Princip je to jednoduchý a na rozdíl od technologie Web Sockets nevyžaduje žádné speciální softwarové vybavení na straně serveru. To je dle mého názoru největší výhodou této technologie. Další výhodou hrající ve prospěch AJAXu je velká podpora mezi prohlížeči. Pokud se ošetří určité situace, dá se úspěšně tvrdit, že je AJAX funkční ve všech dnešních prohlížečích.

[4] Obrázek je dostupný na adrese: http://www.w3schools.com/ajax/ajax_intro.asp

Technologie AJAX je velice komplexní a úplné popsání této technologie by samo vydalo na samostatnou bakalářskou práci. Tato technologie byla již nejednou zpracovaná, a proto jen shrnu to nejnutnější pro praktickou část mé práce. Základem práce je objekt XMLHttpRequest. Tento objekt obsahuje metody pro jako open(), send() či setRequestHeader(). Dále tento objekt zahrnuje události jako onreadystatechange, na níž lze napojit funkci s kódem, který se vykoná po jejím vyvolání. Nakonec také objekt obsahuje proměnné jako responseXML či responseText, v nichž jsou vrácená data obsažena. Základní princip je takový, že po načtení dokumentu vytvořím instanci objektu XMLHttpRequest. Ve chvíli, kdy potřebuji provést asynchronní dotaz na server, zavolám metodu open(). Jejími parametry jsou "druh přenosu dat", "adresa ke vzdálenému skriptu" a pak logická hodnota true nebo false, která určuje, jestli se má dotaz provést asynchronně či synchronně. Pokud se provede dotaz synchronně, znamená to, že se čeká v běhu programu na odpověď a dokud ta nepřijde, tak se nemůže pokračovat v prováděném kódu.

Jak jsem již zmiňoval v předchozím textu, aby byla zajištěna podpora mezi všemi dnes používanými prohlížeči, mezi které se stále bohužel řadí i prohlížeče IE5 a IE6, je nutné při vytváření objektu XMLHttpRequest zakomponovat podmínku, která zohlední i tyto prohlížeče. Kód by mohl vypadat následovně ([12]):

```
var xmlhttp;
if(window.XMLHttpRequest){
 xmlhttp = new XMLHttpRequest();
}
else {
 xmlhttp=ActiveXObject("Microsoft.XMLHTTP";
}
```

Po takovéto podmínce bude AJAX fungovat i ve zmiňovaných prohlížečích, ovšem pouze pod podmínkou, že je v nich povoleno spouštění ActiveX rozšíření. Jednoduchý příklad takové Ajaxové komunikace by mohl vypadat podobně jako v trochu poupraveném příkladu z w3schools.com, který naleznete v příkladu **07_ajax**([12]):

```
var xmlhttp;
if(window.XMLHttpRequest){
 xmlhttp = new XMLHttpRequest();
}
else {
 xmlhttp =ActiveXObject("Microsoft.XMLHTTP");
}
// funkce pro získání dat
function getData(){
 xmlhttp.open("GET","ajax_info.txt",true);
 xmlhttp.send();
}
// registrace události pro zpracování dat
xmlhttp. xmlhttp.onreadystatechange=function(){
 if(xmlhttp.readyState==4 &&
 xmlhttp.status==200)
 {
 document.getElementById( "test2" ).innerHTML =
xmlhttp.responseText;
 }
}
```

```
var tlacitko = document.getElementById(
           "tlacitko" );

// registrace události kliknutí na tlačítko

addEvent(tlacitko, 'click', function(){

 getData();

});
```

Z tohoto příkladu lze vyvodit hned několik závěrů, a to:

1. Odpovědí ze serveru je vše, co skript na adrese volané ve funkci open u objektu XMLHttpRequest zobrazí na standardním výstupu (v našem příkladě se jedná o textový soubor, ale stejně tak dobře by se mohlo jednat o nějaký serverový skript generující dynamický obsah dle vstupů).

2. Díky asynchronnímu programování funkce musíme programovou logiku rozdělit do bloků a ty napárovat na události, při kterých se mají dané bloky provádět.

Při vyvolání události onreadystatechange na objektu XMLHttpRequest se kontrolují stavy v proměnné readyState a v proměnné status. Použití těchto stavů umožňuje programátorovi snadno ošetřit chybové stavy při přenosu informací.

Proměnné readyState a status můžou nabývat následujících stavů ([12]):

Vlastnost	Popis
onreadystatechange	Obsahuje funkci (nebo název funkce), která je volána automaticky pokaždé, kdy dojde ke změně vlastnosti readyState.
readyState	Hodnoty statusů, které nabývají v objektu XMLHttpRequest. Změny od 0 do 4: 0: žádost není inicializovaná 1: připojení k serveru založeno 2: žádost přijata 3: žádost se vyřizuje 4: žádost byla dokončena a čeká se na odpověď
status	200: vše v pořádku 404: Stránka nenalezena

Poslední důležitou metodou objektu XMLHttpRequest, bez které by se má praktická část neobešla, je metoda setRequestHeader(). Tato metoda umožňuje nastavení formátu hlavičky dat, jenž budu odesílat. Metoda musí být zavolána po metodě open(), ale ještě před metodou send().

Tím by byla popsána cesta dat ze serveru do prohlížeče. Ovšem ještě zbývá dostat data z prohlížeče k serveru. k tomu lze použít dvě metody. První možností je metoda GET. Data se při této metodě nastavují přímo do adresy stejně, jako je tomu u obyčejných odkazů. Nevýhoda této metody spočívá v omezení počtu znaků adresy. Metoda GET je mnohem méně robustní než druhá metoda POST, ale je rychlejší a jednodušší než POST. Následující kód může sloužit jako ukázka odeslání jednoho parametru jménem název s hodnotou test:

```
xmlhttp.open("GET","test.php?nazev=test,
            true);
```

Stejná data by se odeslala pomocí metody POST následovně:

```
xmlhttp.open("POST","test.php",true);

xmlhttp.setRequestHeader("Content-
type","application/x-www-form-urlencoded");

xmlhttp.send("nazev=test");
```

Následující data budou odeslána tak, jako by odešly z formuláře.

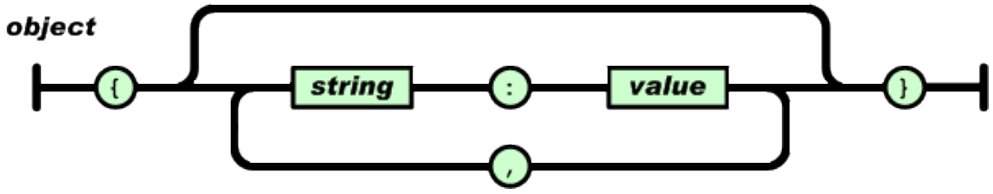
2.2.3 Datové typy pro komunikaci

V předchozí podkapitole bylo vysvětleno, jak lze provádět asynchronní komunikaci mezi prohlížečem a serverem. Data, která odesílá serverový skript, mohou být poslána jako XML nebo jako text. XML má jednoznačnou výhodu v rozšířenosti, ale trpí jedním neduhem. Dle mého názoru je nejslabším místem formátu XML zdvojení tagů. Pokud chci přenést informaci, musím k ní přidat ještě navíc element, kterým ty konkrétní informace obalím. Tento název elementu je navíc díky syntaxi XML nutné přenést 2x. Jednou jako start elementu a podruhé jako ukončení elementu.

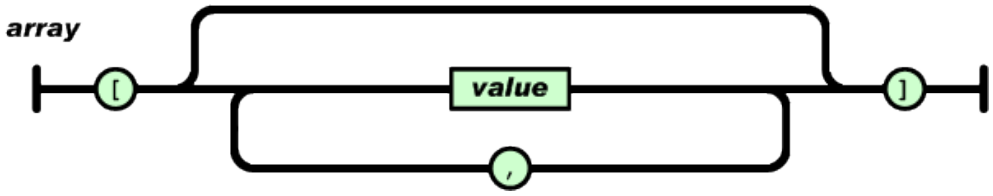
Pokud ovšem tato režie pro formát XML začne být neúnosná, jak se dá srovnat? Odpovědí je JSON formát. Formát JSON vystihuje nejlépe definice na stránkách tvůrců.

JSON (JavaScript Object Notation) je úsporný formát pro výměny dat. Je snadno čitelný i zapisovatelný pro lidi. Stroje ho snadno dekódují i generují ([13]). Dále je založen na jazyku JavaScript a Standard ECMA-262 třetí generace a podporovaný jazyky rodiny C, Java, Perl, Python nebo také jazykem PHP. Syntaxi tohoto formátu asi nejlépe popíší následující obrázky[5]:

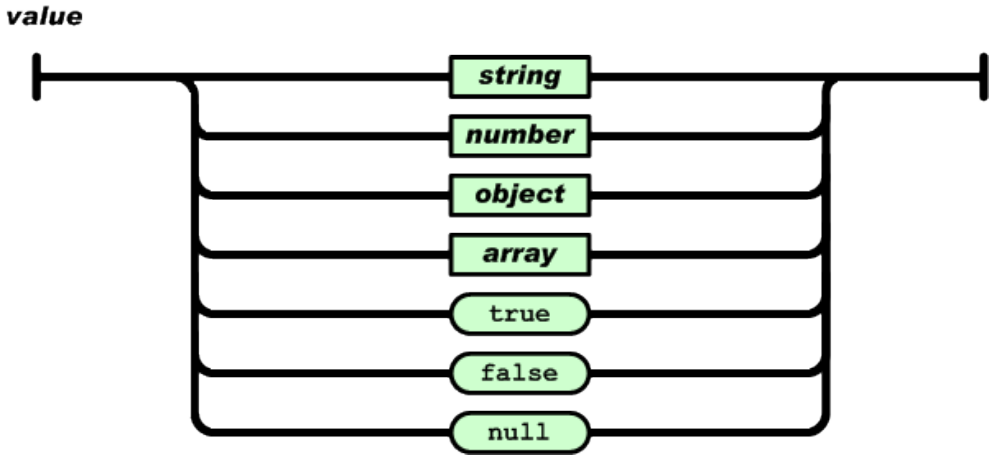
[5] Obrázky jsou dostupné z adresy www.json.org



Obr. 3 Zápis objektů



Obr. 4 Zápis polí



Obr. 5 Zápis hodnot

Přepis takového pole by pak mohl vypadat např. takto:

```
// v XML
<0>
 <0>test 1</0>
 <1>
      <0>test 2.1</0>
      <1>test 2.2</1>
 </1>
</0>
// v jazyce JSON
['test 1', ['test 2.1', 'test 2.2']]
```

Úspora přenesených dat za pomoci JSON oproti XML je vysoká. Bez ohledu na data samotná je syntaktická režie u JSON 12 znaků, zatímco u XML 32 znaků. Záměrně bylo použito co nejmenších názvů elementů u XML, tak aby se neztratila informace. I přes tyto krátké názvy je syntaktická režie u tohoto drobného příkladu více než dvojnásobná. Z tohoto důvodu použiji v mé praktické práci pro přenos dat formát JSON.

2.3 DND Upload souborů

Na začátku zpracovávání této práce jsem přemýšlel, k jakému účelu bude sloužit tato kapitola. Nikomu přeci nemůže vadit kliknutí na tlačítko pro výběr souboru a možnost nahrát fotku na server. Každý to přeci zná a používá se to všude. Neexistuje tedy důvod, proč to měnit. Pak jsem shodou okolností přešel na platformu počítačů od firmy APPLE. Při této migraci jsem si všiml jedné poměrně užitečné vlastnosti operačního systému Mac OS X a to, že kompletně všechny vstupy jsou připraveny na ovládání pomocí DND. Když jsem přemýšlel, jak vypálit obraz na CD, viděl jsem tam dvě textová pole bez známého tlačítka otevřít soubor. v tu chvíli mě napadlo chytnout obraz a přetáhnout ho do kolonky zdroj. Do kolonky cíl sem poté přetáhl mechaniku, kam sem chtěl obraz vypálit. To bylo vše, jak prosté. Proč jsem si ve své práci neodpustil tento krátký osobní příběh? Jedno rčení praví: "V jednoduchosti je krása". a co může být jednodušší než chytnout konkrétní soubor a přetáhnout ho do stránky. Co ale v případě, když chci nahrát více souborů? Označím více souborů a přetáhnu je.

Dnes je tato funkce již plně integrována v emailovém klientu firmou Google. Pokud přetáhnu nad stránku nějaký soubor, přibude ve formuláři pro vytvoření nového emailu nový blok. Do toho je možné pustit tažený soubor a nahrát tak soubor jako přílohu. Krásná funkcionalita, kterou používám zcela běžně. Jak ovšem taková technika funguje? To se pokusím v rychlosti nastínit v této kapitole.

2.3.1 Nahrávání souborů do prohlížeče

Pro nahrávání souborů se dlouhou dobu používalo elementu input typu "file". Výhodou této techniky je, že jí bez problémů podporují všechny prohlížeče. Proto je správné, aby nové techniky pro upload byly rozšířením původní metody. v přiloženém příkladu **08_upload**, který vychází z příkladu na DND upload

napsaném ve frameworku jQuery[6], budu vycházet v následujících příkladech. Kód je přepsán do čistého HTML5 a poupraven pro mé potřeby názorné demonstrace principu HTML5 DND UPLOAD API.

Základem DND uploadu souborů jsou stejně jako u obyčejného HTML5 DND API události **dragover**, **dragleave** a **drop**. Zatímco si dragover a dragleave najde uplatnění především u ovlivňování visuální stránky aplikace, událost drop se provede při uvolnění souborů nad reagující zónou. Jak je vidět na následujícím kódu, při najetí tažené fotografie do reagující zóny se nastaví na reagujícím bloku třída hover. Ta slouží jako zaobalení sady stylů. Možná Vás napadá, proč jsem nepoužil subtřídu hover. To by ale v tomto případě nefungovalo. Subtřída hover reaguje pouze na událost najetí kurzoru nad danou oblast prvku, ovšem ne v momentě, kdy je stlačené tlačítko.

```
addEvent(dropzone, 'dragover', function(event){
 dropzone.classList.add('hover');
 return false;
});
addEvent(dropzone,'dragleave', function(event){
 dropzone.classList.remove('hover');
 return false;
});
```

Pro nastavení či odebraní CSS třídy u elementu bylo použito vlastnosti classList. Ta umožňuje snadnou práci s atributem class každého HTMLElementu. Vlastnost obsahuje metody add, remove, toggle a contains. Bohužel tuto užitečnou vlastnost obsahují pouze moderní prohlížeče s podporou HTML5. Nefunkčnost ve starších prohlížečích jako IE 9 a starší lze opravit "HACKem", který tuto funkcionalitu do prohlížeče přidá[7].

Data fotografií jsou dostupná v objektu typu DragEvent uvnitř vlastnosti dataTransfer.files. Tento objekt je vytvořen při vyvolání události drop a nastaven

[6] Původní příklad napsaný ve frameworku jQuery je dostupný z adresy: http://www.inwebson.com/html5/html5-drag-and-drop-file-upload-with-canvas/

[7] Kód "HACKu" pro doplnění podpory vlastnosti classList je dostupný z adresy: https://developer.mozilla.org/fr/docs/DOM/HTMLInputElement

jako první parametr funkce, která se vykonává při vyvolání události. v přiloženém příkladě **08_upload** je vytvořen ukazatel files a zavolána funkce processFiles, která má za úkol provést zpracování souborů. v této funkci se provádí kontrola počtu souborů a kontroluje se podpora FileReader API. Toto API je podporováno všemi moderními prohlížeči, což dokládá i Obr. 6[8]

FileReader API - Working Draft

Global user stats*:
Support: 62.35%

Method of reading the contents of a File or Blob object into memory

Resources: FileReader API

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	Opera Mobile	Chrome for Android	Firefox for Android
21 versions back			4.0									
20 versions back			5.0									
19 versions back		2.0	6.0									
18 versions back		3.0	7.0									
17 versions back		3.5	8.0									
16 versions back		3.6	9.0									
15 versions back		4.0	10.0									
14 versions back		5.0	11.0									
13 versions back		6.0	12.0									
12 versions back		7.0	13.0									
11 versions back		8.0	14.0									
10 versions back		9.0	15.0		9.0							
9 versions back		10.0	16.0		9.5-9.6							
8 versions back		11.0	17.0		10.0-10.1							
7 versions back		12.0	18.0		10.5							
6 versions back		13.0	19.0		10.6			2.1				
5 versions back	5.5	14.0	20.0	3.1	11.0			2.2		10.0		
4 versions back	6.0	15.0	21.0	3.2	11.1	3.2		2.3		11.0		
3 versions back	7.0	16.0	22.0	4.0	11.5	4.0-4.1		3.0		11.1		
2 versions back	8.0	17.0	23.0	5.0	11.6	4.2-4.3		4.0		11.5		
Previous version	9.0	18.0	24.0	5.1	12.0	5.0-5.1		4.1		12.0		
Current	10.0	19.0	25.0	6.0	12.1	6.0	5.0-7.0	4.2	7.0	12.1	25.0	19.0
Near future		20.0	26.0		12.5				10.0			
Farther future		21.0	27.0									

Obr. 6 Podpora FileReader API

Bohužel opět je API podporováno v prohlížeči Internet Explorer až od verze 10. Ostatní prohlížeče s ním již nemají problém.

Pole obsahující soubory se zpracovávají v cyklu a pro každý soubor je zavolána funkce readFile. Tato funkce provede kontrolu konkrétního souboru (zda se jedná o obrázky) a vytvoří instanci objektu typu FileReader. Tento objekt obsahuje několik metod, které umožňují načtení souboru. Soubor je možné načíst metodou readAsDataURL(in Blob file). Parametrem této metody je ukazatel na data, která vrací dialogové okno po výběru souboru nebo na která ukazuje ukazatel dataTransfer.files v objektu typu DragEvent události drop. Dalšími možnostmi načtení obrázku jsou metody readAsBinaryString, readAsArrayBuffer a readAsText[9].

[8] Obrázek je dostupný z adresy: http://caniuse.com/filereader

[9] Bližší informace o metodách jsou dostupné z adresy: https://developer.mozilla.org/en-US/docs/DOM/FileReader

Tyto metody začnou načítat soubor z vašeho počítače do prohlížeče. Protože soubory můžou být relativně velké a načítání obrázku může trvat poměrně dlouho, objekt FileReader obsahuje událost onload. Tato událost se vyvolá, jakmile je nahrávání souboru(ů) kompletní. Proto se nemusí zastavit běh skriptu, dokud se soubor nenahraje, ale celé nahrávání proběhne asynchronně. Výsledná data obrázku jsou dostupná v objektu DragEvent ve vlastnosti target.result. Taková data můžeme dále zpracovávat jako v našem příkladě, kde jsem vytvořil nový element img, do atributu src jsem vložil data obrázku a element vložil do bloku target.

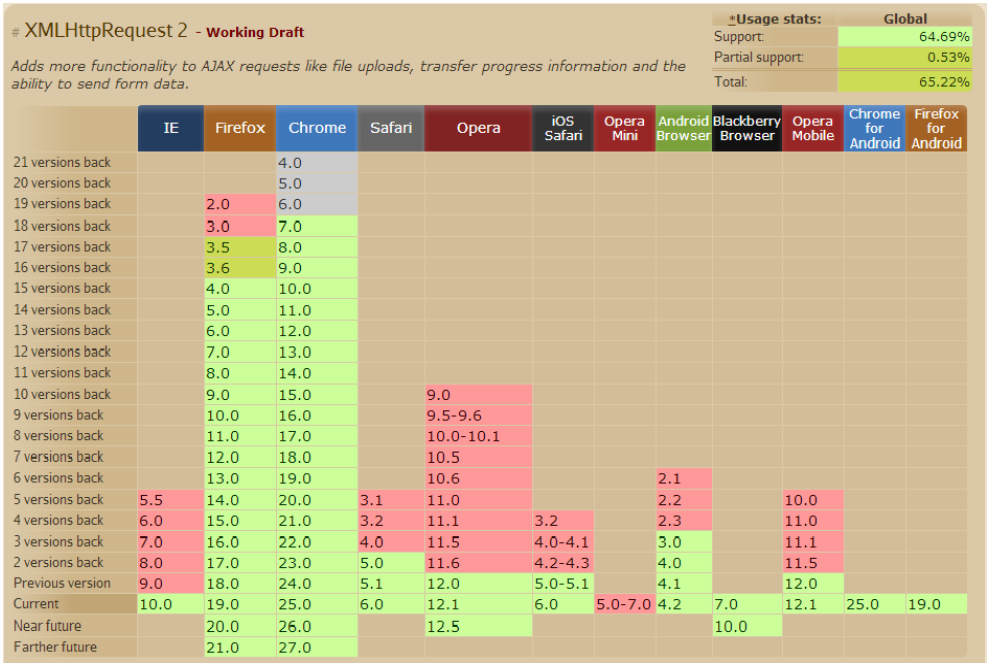
2.3.2 Generování náhledu

Příklad **08_upload** není pro praktické použití příliš ideální, protože používá fotografii v originální velikosti. To v případě velkých fotografií může znamenat zdlouhavé a na datovou komunikaci náročné načítání. Dříve se takový problém řešil vygenerováním náhledu na straně serveru serverovými skripty. To sebou neslo jeden zásadní problém. Každé vygenerování náhledu nezanedbatelně vytěžovalo server. Středně velká fotografie ve formátu jpg po dekompresi může zabírat v operační paměti serveru i několik desítek megabytů. To v případě uploadu několika fotografií současně může plně vytížit i velice výkonný server. Logickým krokem tedy je, přesunout tuto operaci na stranu klienta a vygenerovaný obrázek včetně náhledu pouze nahrát na server. Znamená to sice větší datovou komunikaci směrem od klienta k serveru, ale ušetří to nemalý výpočetní výkon.

Zpracování fotografie v prohlížeči bude řešeno pomocí elementu canvas. Popis všech možností tohoto elementu je nad rámec této práce, proto pouze ukážu, jak takový náhled vygenerovat. Jak je možné vidět v přiloženém příkladu **09_thumb**, upravil jsem funkci readFile. v momentě, kdy dojde k načtení originálního obrázku, začnu generovat náhled. Toto generování se provádí funkcí getThumbDataUrl(original, maxWidth) a vygeneruje datovou url obrázku o šířce dle druhého parametru. Základní logika této techniky spočívá ve vytvoření elementu canvas o velikosti požadovaného náhledu a na něj je vykreslen obrázek. Nakonec se provede export plátna na datovou url. Z této url je vytvořen nový element obrázku a zapouzdřen v odkazu směřujícím na originální obrázek. Odkaz se poté vloží do bloku result. Tento příklad již začíná vypadat jako opravdu jednoduchá fotogalerie.

2.3.3 Upload souborů na server

Předchozí příklad **09_thumb** nám vygeneroval z originálního souboru náhled a vložil ho do stránky. v následujícím příkladu **10_upload_ajax** ukážu, jak přenést tyto vygenerované obrázky na server. Příklad **10_upload_ajax** přímo vychází z příkladu **09_thumb**. Na tomto příkladě demonstruji dva různé způsoby přenášení souborů pomocí technologie AJAX. Základem nahrávání souborů je opět objekt XMLHttpRequest, přesněji ve verzi 2. Tento objekt podporují všechny dnešní moderní prohlížeče[10].



XMLHttpRequest 2 - Working Draft

Adds more functionality to AJAX requests like file uploads, transfer progress information and the ability to send form data.

*Usage stats:	Global
Support:	64.69%
Partial support:	0.53%
Total:	65.22%

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	Opera Mobile	Chrome for Android	Firefox for Android
21 versions back			4.0									
20 versions back			5.0									
19 versions back		2.0	6.0									
18 versions back		3.0	7.0									
17 versions back		3.5	8.0									
16 versions back		3.6	9.0									
15 versions back		4.0	10.0									
14 versions back		5.0	11.0									
13 versions back		6.0	12.0									
12 versions back		7.0	13.0									
11 versions back		8.0	14.0									
10 versions back		9.0	15.0		9.0							
9 versions back		10.0	16.0		9.5-9.6							
8 versions back		11.0	17.0		10.0-10.1							
7 versions back		12.0	18.0		10.5							
6 versions back		13.0	19.0		10.6			2.1				
5 versions back	5.5	14.0	20.0	3.1	11.0			2.2		10.0		
4 versions back	6.0	15.0	21.0	3.2	11.1	3.2		2.3		11.0		
3 versions back	7.0	16.0	22.0	4.0	11.5	4.0-4.1		3.0		11.1		
2 versions back	8.0	17.0	23.0	5.0	11.6	4.2-4.3		4.0		11.5		
Previous version	9.0	18.0	24.0	5.1	12.0	5.0-5.1		4.1		12.0		
Current	10.0	19.0	25.0	6.0	12.1	6.0	5.0-7.0	4.2	7.0	12.1	25.0	19.0
Near future		20.0	26.0		12.5				10.0			
Farther future		21.0	27.0									

Obr. 7 Podpora objektu XMLHttpRequest 2 mezi prohlížeči

Ve funkci readFile() se po vygenerování náhledu volá funkce uploadOriginalToServer. Jejími parametry je původní soubor přetažený do prohlížeče a datová URL náhledu. Pro samotné předání dat se může použít objekt FormData, který už je opět podporovaný všemi moderními prohlížeči. Tento objekt umožňuje snadné vytváření formulářových dat v JavaScriptu. Přidání vstupů je realizováno metodou append, která má dva parametry. První parametr představuje řetězec sloužící pro pojmenování klíče vstupu a druhý hodnotu vstupu. Další krok spočívá ve vytvoření instance objektu XMLHttpRequest, který je otevřen na adresu script.php. Metodou send() se odešle instance objektu FormData, kterou jsem v předchozím kroku naplnil daty. Poslední věcí, kterou je třeba zajistit, je nastavení

[10] Obrázek Obr. 7 je dostupný z adresy: XMLHttpRequest2. Http://caniuse.com [online]. 2012 [cit. 2013-03-14]. Dostupné z: http://caniuse.com/#feat=xhr2

funkce na událost onload. Tato událost se vyvolá po dokončení nahrávání souboru na server. To může v případě rozměrných fotografií trvat poměrně dlouhou dobu, a tak je výhodnější to řešit asynchronním voláním funkce. v momentě, kdy byla úspěšně nahrána velká fotografie, se začne nahrávat vygenerovaný náhled. Zde se použije druhý způsob nahrávání dat na server. v předchozím příkladě jsme si vygenerovali náhled ve formátu datové URL. Ten bude odeslán uvnitř funkce **uploadThumbToServer()**. Tato funkce má 2 povinné parametry, a to ukazatel na původní soubor a náhled ve formátu datové URL. Příprava dat je opět řešena za pomoci objektu FormData a odeslána AJAXem na serverový skript script.php. Jakmile je soubor úspěšně nahrán, tak se vytvoří nový element odkazu, který již směřuje na obrázky uložené na serveru.

Jak bylo zmíněno v předchozím odstavci, data jsou odesílána na serverový skript napsaný v jazyce PHP. Když se v rychlosti podíváme, co je jeho obsahem, tak zjistíme, že se na začátku skriptu vytváří ukazatel na soubor **output.txt**. Ten je zde pouze pro možnost snadného ladění programu, protože v případě skriptů volaných AJAXem nemáme možnost zobrazit si standardní výstup. Protože je skript použit pro oba dva přístupy nahrávání souborů, vytvoří se zde podmínka, která rozhodne, zda-li nahrávám originální soubor nebo datovou URL. v prvním případě se pouze zavolá php funkce move_uploaded_file, která překopíruje fotku do nahrávací složky na serveru.V druhém případě se kontroluje datový typ odeslaných dat datové URL. Pokud se jedná o formát obrázků, provede se odseparování hlavičky dat php metodou substr, a poté se zakódovaná data dekódují. Dekódovaná data se pak uloží do souboru.

2.3.4 Sledování průběhu nahrávání

V příkladu **11_upload_progres** se zabývám vytvořením ukazatele stavu nahrávání fotografie na server. v HTML5 byl přidán nový element **progress,** který je přímo určen k ukazování průběhu různých akcí. Příklad byl navržen tak, aby po začátku nahrávání přibyl v bloku **droparea** tento element a průběžně se ukazovala aktuální hodnota nahrávání. Po dokončení nahrávání se progress bar smaže a vypíše se hláška o dokončení nahrávání.

V objektu XMLHttpRequest level 2 byla přidána u vlastnosti upload událost progress. Musím říci, že tato vlastnost mi ukázala záludnost zpracovávání

technologie, která je stále ve vývoji. Nespočet příkladů dostupných na internetu uvádí v postupu pro registraci této události napojení přímo na objekt XMLHttpRequest. To je pravděpodobně pozůstatek z nějaké rané verze specifikace, který v nejnovějších verzích prohlížečů nefunguje. Nakonec mi pomohlo jen několikahodinové hledání chyby metodou pokus omyl. Tím se dostávám k vysvětlení jinak vcelku bezproblémového kódu. Ačkoli se všechny události používané při AJAXovém přenosu dat nastavují přímo na objektu XMLHttpRequest, tak událost pro sledování průběhu nahrávání se registruje následovně:

```
xhr.upload.addEventListener('progress',
                    function (event){
 if(event.lengthComputable){
      // kód pro výpočet přenesených dat
 }
 else {
      // kód pro chybu
 }
}, false);
```

Vlastnost objektu XMLHttpProgressEvent event.lengthComputable obsahuje logickou hodnotu, která ukazuje, zda je možné data počítat. Pokud je hodnota rovna false, pak jsou i vlastnosti používané pro výpočet nastaveny na hodnotu 0. v opačném případě se může provést výpočet. Vlastnosti nutné pro výpočet jsou **event.loaded** a **event.total**. Přičemž ta první obsahuje informaci o počtu již přenesených dat v bytech a ta druhá obsahuje celkovou velikost souboru. Pokud se tyto dvě hodnoty vydělí a výsledek je vynásoben 100, po zaokrouhlení funkcí Math.round na celá čísla získáme procentuální stav nahrávání. Ten se poté nastaví do atributu value u elementu progress. v rámci zpětné kompatibility lze zobrazit dovnitř textové hodnoty elementu ještě textovou zprávu se stavem. Tato zpráva je zobrazena v prohlížečích bez podpory nových HTML5 elementů jako např. IE9 a starší. Ve funkci prováděné při události **load** se poté odebere element progress a do bloku s identifikátorem progress se vloží textová informace o dokončení nahrávání.

2.4 Formátování fotografie

Při psaní mé práce jsem se velice rozhodoval, zda se tato kapitola opravdu do práce hodí. Webová aplikace na úpravu fotografií by s přehledem vydala na několik diplomových prací a s mým téma HTML5 DND API toho také příliš společného nemá. Na druhou stranu element canvas umožňuje ovlivňování scény táhnutím jednotlivých objektů ve scéně. Proto jsem se rozhodl pro kompromis. v této kapitole ukážu jednoduché operace s objektem canvas, které jsou prováděny táhnutím myši nebo ty, o kterých si myslím, že by se do fotogalerie hodili.

2.4.1 Tvorba výřezu

Tvorba výřezu fotografie je přesně tou funkcí, která mi nejvíce chyběla ve většině fotogalerií. S příchodem frameworků jako jQuery se i tato funkcionalita začíná čím dál častěji objevovat na webu, ale stále to není dokonalé. Proč má člověk šahat po grafickém software, když si chce jen trochu poupravit danou fotografii.

Na příkladě **12_uprava_foto_vyrez** je ukázána práce s elementem canvas při DND ovládání. Příklad vychází z tutoriálu HTML5 Canvas Image Crop Tutorial napsaným Erikem Rowellem[11]. v mém příkladu jsem z originálního tutoriálu vycházel z logické stavby programu. Vzhledem k tomu, že tutoriál je napsaný ve frameworku jQuery, tak byl přepsán do čistého HTML5 a bylo opraveno několik nedostatků. Knihovna js/script.js byla přepsána pro mé potřeby a připravena pro obecné použití v různých aplikacích.

JavaScriptová knihovna je načtena v hlavičce HTML dokumentu. Aby bylo zaručeno správné načtení knihovny, inicializační metoda je volána až po úplném načtení dokumentu. To je zaručeno nastavením události **DOMContentLoaded**, která se vyvolá až po úplném načtení celé stránky včetně obrázků. Toto je nová událost z jazyka HTML5, kterou podporují všechny moderní prohlížeče s podporou HTML5. v tělu funkce napojené na tuto událost se zavolá funkce loadImage a nastaví se událost kliknutí na tlačítko pro vytvoření výřezu. Parametry funkce loadImage jsou dva. První je odkaz na obrázek a druhý obsahuje selektor pro identifikaci kreslící plochy canvas.

[11] Originální tutoriál od Erika Rowella ze dne 21. října 2012 je dostupný z adresy: http://www.html5canvastutorials.com/tutorials/html5-canvas-image-crop/

Obsahem knihovny **js/script.js** je objekt typu Selector a přístupová funkce getCropDataURL(). Dále je v knihovně definováno několik globálních proměnných sloužících pro předávání dat mimo objekt. Nemá smysl vysvětlovat každý řádek kódu této knihovny, ale pokusím se shrnout ve zkratce funkci knihovny:

1. Vybere se element canvas
2. nastaví se mu rozměry dle obrázku.
    a. Pokud je obrázek širší než maximální povolená šířka v globální proměnné maxWidthImage, tak se provede automatické zmenšení originálu obrázku na tuto šířku a vygeneruje se v daném poměru výška.
    b. Pokud je obrázek užší, tak se pracuje rovnou s originálem.
3. plátno se při každém překreslení vyčistí od předchozí scény
4. poté se vykreslí fotografie v plném rozměru
5. celá scéna se překreslí průhledným čtvercem, který slouží jako ztmavení
6. do scény se vykreslí aktuálně nastavený výřez z obrázku
7. do globální proměnné maxWidthImage se uloží datová URL aktuálního výřezu
8. vykreslí se rám výřezu spolu s ovládacími prvky

Výše zmíněná posloupnost příkazů se provede při každém zaznamenaném pohybu kurzoru v oblasti elementu canvas. Knihovna byla upravena do abstraktní podoby, aby byla její aplikace co nejuniverzálnější.

2.4.2 Otočení fotografie

Funkci otáčení fotografie jsem demonstroval na příkladu **13_uprava_foto_otoceni**. Tato funkce není nikterak složitá. Základní princip spočívá v použití dvou metod objektu canvas. První metoda **translate(posun_v_x, posun_v_y)** posune nulový bod kreslení v osách x a y. Druhá metoda **rotate(uhel)** otočí směr kreslítka. Po tomto nastavení už stačí jen vykreslit obrázek standardně jako v předchozích příkladech. Parametr metody rotate je úhel otočení v radiánech. Já jsem si zvolil pracovní interval 0π až 2 π. Pokud dojde k převýšení hodnoty 2 π, tak se automaticky nastaví úhel na 0 π. Poslední důležitou věcí je změna poměru stran objektu canvas. Pokud se jedná o úhel π / 2 a 2 / 3 π, tak dochází k prohození šířky plátna za výšku a obráceně. v tomto příkladě se stejně jako v příkladě na výřez

fotografie zohledňuje skutečná velikost fotografie. Pokud je šířka fotografie větší než maximální, nastaví se šířka na tento maximální rozměr.

2.4.3 Efekty fotografie

Posledním příkladem v mé teoretické části práce je ukázka, jak v HTML5 tvořit efekty na úpravu fotografií. Existuje celá řada efektů, které lze snadno implementovat do naší aplikace. v příkladu **14_uprava_foto_efekty** je vytvořeno několik zajímavých efektů a sada textur, které lze kombinovat s jednotlivými efekty. Pokud bych se pokusil vysvětlit logiku filtrů, tak základem je vykreslení obrázku do objektu canvas. Poté je zavolána metoda **getImageData()** objektu typu CanvasRenderingContext2D ctx. Ta vrací objekt typu ImageData, který obsahuje jednotlivé pixely obrázku. Metoda data() objektu ImageData vrací objekt typu Uint8ClampedArray a rozdělí jednotlivé pixely do barevných složek v hodnotách od 0 do 255. Hodnoty pole jsou uloženy po čtveřicích, přičemž první hodnota reprezentuje červenou složku, druhá zelenou, třetí modrou a čtvrtá je alpha kanál. S takto připravenými daty se již dá poměrně snadno pracovat. Pouze se v cyklu projde celé pole (ideálně ve skocích po čtyřech) a ovlivní se pixely podle požadovaného algoritmu. Ovlivněná data se nakonec vykreslí metodou putImageData do objektu ctx. Jednotlivé efekty se liší pouze tělem cyklu. Tyto efekty jsou obecně známé a nemá smysl je podrobněji rozebírat.

3 Test podpory DND API mezi prohlížeči

V této kapitole jsem se zabýval testováním podpory DND API mezi aktuálními prohlížeči. v jedné podkapitole jsem se zabýval výkonem JavaScriptového jádra jednotlivých prohlížečů a v dalších podkapitolách jsem se zaměřil na otestování podpory jednotlivých API a technik v HTML5.

3.1 Test výkonu javascriptového jádra prohlížečů

Při testování efektů bylo poměrně zajímavé pozorovat rozdíly výkonu při generování stejného efektu mezi různými prohlížeči. Z toho důvodu jsem se rozhodl do mé práce doplnit tuto kapitolu, která sice ne přímo souvisí s HTML5 DND API, ale určitě bude pro případného čtenáře zajímavá.

Výkon JavaScriptového jádra mezi moderními prohlížeči jsem v příkladu 15_uprava_foto_efekty_test_vykonu měřil následovně: v cyklu jsem provedl 30 x (u náročných efektů bylo generováno 5x) generování měřeného efektu. Každé generování bylo změřeno s přesností na milisekundy. Aby bylo měření co nejméně ovlivněno, těsně před voláním funkce efektu je uložen aktuální čas v sekundách. Uložení je realizováno funkcí microtime(), která má jeden parametr typu bool. Pokud je hodnota parametru rovna true, funkce vrací aktuální čas v sekundách. v opačném případě funkce vrací aktuální čas v milisekundách. Druhým krokem měření je odečtení změřeného času od aktuálního času. Tím je změřena doba, za kterou bylo generování dokončeno. Každé měření je ukládáno ve formátu CSV textu v bloku results. Po dokončení testování jsem výsledky měření uložil do CSV souboru pro daný efekt. Vždy jsem při testování dodržel pořadí prohlížečů IE9, IE10, FF19, Chrome 25 a Opera 12.

Měření bylo provedeno na jednom stroji (Intel Core I7-3770 3.4-3.8GHz, 16GB DDR3, Windows 7 Ultimate 64 bit). v následující tabulce jsou vidět průměrné časy, za které vygenerovaly prohlížeče efekty na fotografii o rozměrech obrázku 800 x 531 pixelů. Pro maximální přesnost měření byl v momentě testování spuštěn vždy pouze testovaný prohlížeč.

	Čer. fotka	Bar. inverze	Sepia efekt	Efekt přesv.	Threshold efekt	Convolving efekt 1	Convolving efekt 2	Čas všech testů
IE 9	0.53	0.34	3.20	0.33	0.36	4.79	6.12	15.66
IE 10	0.19	0.10	3.32	0.10	0.11	13.00	12.99	29.83
FF 19	0.10	0.09	1.82	0.09	0.10	0.61	0.62	3.43
GC25	0.16	0.14	0.18	0.15	0.16	0.45	0.48	1.71
O 12	0.29	0.23	0.35	0.23	0.23	1.95	1.97	5.24

Legenda:

- IE 9 - Internet Explorer v. 9
- IE 10 - Internet Explorer v. 10
- FF 19 - Mozilla Firefox v. 19
- GC 25 - Google Chrome v. 25
- O 12 - Opera v. 12

V tabulce lze spatřit, že při zpracování stejného skriptu jsou mezi prohlížeči znatelné rozdíly. Největším překvapením pro mě byly výsledky prohlížeče IE10, protože vyšel z testu nejhůře, ale zároveň je skoro 2 x pomalejší než předchozí verze. Výsledky pro mě byly natolik neuvěřitelné, že jsem testování několikrát opakoval, avšak se stejným výsledkem. Prohlížeč IE 10 zvládl vygenerovat všechny testy za takřka 30 s. Na druhé straně v žebříčku je prohlížeč Google Chrome 25, který všechny testy hravě zvládnul za 1,71 s a zaslouží si tak veliký obdiv. Jen tak pro zajímavost Google Chrome 25 zvládl vygenerovat všechny efekty (průměrné hodnoty z měření každého efektu) rychleji než ostatní prohlížeče jeden ze složitých filtrů. To je dle mého názoru opravdový důkaz kvality tohoto prohlížeče.

3.2 Definování testovacích příkladů

Pro testování funkčnosti HTML5 DND API a dalších funkcionalit jsem využil příkladů, které jsem si připravil v rámci teoretické části mé práce. v této kapitole budu testovat podporu následujících funkcionalit:

- táhnutí elementem div v rámci jedné stránky
- táhnutí elementem div mezi dvěma stránkami
- přetáhnutí souboru z PC do prohlížeče
- nahrávání souborů na server za pomoci XMLHttpRequest 2
- tvorba výřezů za pomoci elementu canvas
- aplikace efektů za pomoci elementu canvas

Táhnutí elementem div v rámci jedné stránky i mezi dvěma stránkami bylo testováno na příkladě **05_drop**. Přetáhnutí souboru z PC do prohlížeče jsem testoval na příkladu **08_upload**. Nahrávání souborů na server za pomoci XMLHttpRequest 2 bylo testováno na příkladě **10_upload_ajax**. Tvorba výřezů byla testována na příkladě **12_uprava_foto_vyrez**. Testování aplikace efektů na fotografii byla testována na příkladě **14_uprava_foto_efekty**.

3.3 Průběh testu napříč prohlížeči

Během testování byla zjišťována podpora dané funkcionality metodou funguje / nefunguje. Prohlížeč buď danou funkcionalitu plně umožňuje a je v tabulce výsledků označen jako podporováno, nebo nefunguje či nefunguje úplně a je označen jako nepodporováno. Veškeré testy byly prováděny na stejné sestavě jako test výkonu JavaScriptového jádra. Výsledky testování jsou vidět v následující tabulce.

		Testované funkcionality					
		1)	2)	3)	4)	5)	6)
Testované prohlížeče	IE 8	✗	✗	✗	✗	✗	✗
	IE 9	✗	✗	✗	✗	✓	✓
	IE 10	✓	✓	✓	✓	✓	✓
	FF 3.6	✓	✓	✓	✗	✓	✗
	FF 19	✓	✓	✓	✓	✓	✓
	Chrome 13	✗	✗	✓	✗	✓	✓
	Chrome 25	✓	✓	✓	✓	✓	✓
	Opera 12	✓	✓	✓	✓	✓	✓

Podporováno ✓
Nepodporováno ✗

Testy:
1) táhnutí elementem div v rámci jedné stránky
2) táhnutí elementem div mezi dvěma stránkami
3) přetáhnutí souboru z PC do prohlížeče
4) nahrávání souborů na server za pomoci XMLHttpRequest 2
5) tvorba výřezů za pomoci elementu canvas
6) aplikace efektů za pomoci elementu canvas

3.4 Vyhodnocení podpory DND API

Z výsledků vyplývá, že pokud chceme v naší aplikaci využívat výhod HTML5, musíme použít nejnovější prohlížeč. Z prohlížečů od firmy Microsoft je použitelná pouze nejnovější verze 10, ale je třeba myslet na výsledek měření výkonu. Dle mého názoru je výkon natolik žalostný, že mi nezbývá než tento prohlížeč nedoporučit. Naopak prohlížeče Mozilla Firefox jsou celkem použitelné již od raných verzí. Ideálním prohlížečem v současné době je dle mého názoru prohlížeč Google Chrome. Nejen, že má v současnosti nejlepší podporu nových technologií, ale vede s přehledem i po stránce výkonu ve zpracování kódu. U tohoto prohlížeče platí, že čím novější verze, tím se dá čekat lepší výkon. Jako poslední byl testován prohlížeč Opera, který se řadí spíše do šedého průměru. Výkon je průměrný a podpora v nejnovější verzi je dobrá. Někdy se tento prohlížeč chová poněkud zvláštně. Nepodporuje například nastavení obrázku pro táhnutý objekt, ale to považuji za chybu spíše kosmetickou než funkční.

Během testování jsem zjistil jedno zvláštní chování při události dragEnter. Všechny prohlížeče kromě těch postavených na jádru WebKit obsahují v současnosti jednu chybu. Pokud vstupujete do kolizní zóny reagujícího objektu z horní strany, událost se neprovede pouze jednou, ale třeba i několikrát za sekundu. Z jiných směrů to je v pořádku. Osobně si myslím, že takovýchto podivností bude v těchto prohlížečích mnohem více, což by vysvětlovalo, proč je rozdíl mezi výkonem prohlížečů postavených na jádře WebKit a těmi ostatními tak velký.

4 Vývoj aplikace

Ve 4 kapitole se zabývám praktickou částí mé práce. Při této části vývoje aplikace byla na základě provedené analýzy navržena logika fungování aplikace. Nemělo by smysl popisovat program řádek po řádku, ale pouze vysvětlím klíčové části programu a popíšu můj vývoj aplikace. Co se povedlo a co ne.

4.1 Návrh fotogalerie

Aplikace byla navržena jako univerzální JavaScriptová fotogalerie s důrazem na jednoduchou implementaci do projektů druhých stran. v první fázi vývoje jsem se pokusil uvědomit si, co mají všechny fotogalerie společné a čím se naopak liší. Uvědomil jsem si, že všechny galerie ukládají informace o cestě k souboru na serveru do databáze a to do různě pojmenovaných sloupců. Dále se ukládá cesta k souboru náhledu a individuálně ještě pořadí, v jakém se mají fotografie vypisovat. Proto bylo třeba vyřešit tuto diverzitu v názvech sloupců. To se mi podařilo vyřešit za pomocí dvou polí v konfiguračním souboru. První pole obsahuje prvky organizované následujícím způsobem. Index prvku je jednotným pojmenováním sloupce dle funkce jakou plní. Hodnota prvku pak nabývá hodnot true nebo false, a to v závislosti na podpoře daného sloupce v rámci systému. Druhé pole potom obsahuje prvky se stejnými indexy jako první pole, ale hodnotami je buď prázdný řetězec, nebo název sloupce používaný v konkrétním systému. Touto elegantní cestou je jednoznačně provedené mapování konkrétní databázové struktury. Databáze je pak již jen obohacena o systémové sloupce jako např. position_x nebo position_y.

4.2 Definice funkcionalit fotogalerie

Jako první bylo třeba definovat funkcionality, které bude aplikace umožňovat. Při definování funkcionalit jsem se snažil zařadit takové funkce, které mi u jiných fotogalerií chyběly. Aplikace musí umožňovat přesouvání fotografií ve fotogalerii, a to jednotlivě po fotografiích nebo hromadně po řádcích a po sloupcích. Dále aplikace musí umožňovat automatické generování náhledů a musí umožňovat jednoduchou editaci fotografie. Mezi editační nástroje patří nástroj výřezu části fotografie, nástroj transformace - zrcadlení(horizontální / vertikální) nebo otočení. Dále by měla aplikace umožňovat aplikovat na fotografii několik efektů.

4.3 Návrh jádra aplikace

Aplikace se připojuje do jakékoli internetové stránky napsané v jazyce HTML5 připojením skriptu administration.js a podpůrných knihoven. Tím je zaručeno, že se do označeného bloku se vygeneruje administrace nebo prezentační část fotogalerie. Jakýkoli požadavek na změnu pozice či úpravy fotografie se provádí asynchronně pomocí AJAX komunikace na serverový skript. Přemýšlel jsem, jak zajistit co nejuniverzálnější přístup k datům a proto jsem navrhnul tzv. DCL rozhranní.

DCL(Data Communication Layer) je mnou navrženou abstraktní vrstvou usnadňující a standardizující obousměrnou komunikaci mezi aplikací a serverem. Veškerá komunikace mezi aplikací a serverem probíhá přes komunikační bránu AJAXComunicationDataGate, na kterou se aplikace v případě potřeby dotazuje. Skript akceptuje pouze data ve formátu DCLData (individuálně zakódované ve formátu JSON) a zároveň je veškerý výstup ze souboru opět ve formátu DCLData a odeslaný metodou POST v proměnné DATA_COMMUNICATION. Tím je zaručena univerzalita komunikace. Předpokládejme, že bude chtít použít tuto aplikaci vývojář programující na platformě .NET nebo Java. Díky výše zmiňované struktuře programu si programátor bude muset naprogramovat pouze serverový skript AJAXCommunicationDataGate. Díky specifikaci DCLData programátor nemusí přemýšlet nad tím, jak je co řešeno na straně klienta. Pouze uloží data a vygeneruje odpověď.

Formát DCLData je navržen jako pole o čtyřech prvcích. První prvek s indexem QueryType definuje druh dotazu. Druhý prvek pole s indexem QueryState označuje stav zprávy. QueryState rovno hodnotě 100 reprezentuje, že odesílatel nezjistil žádný problém. Pokud je stav menší než 100, tak se vyskytl nějaký problém o kterém je uložena informace ve třetím prvku pole s indexem QueryMessage. Poslední prvek s indexem DataArray obsahuje pole polí. Každý prvek pole DataArray obsahuje 2 vnořené prvky. Jeden prvek je s indexem DataName a reprezentuje název předávané proměnné, zatímco druhý je s indexem DataValue a obsahuje hodnotu dat. Takto definovanou strukturou lze odeslat jakákoliv data bez ztráty informace.

Hlavním objektem klientské části aplikace je objekt CORE, který zajišťuje načtení knihoven a nastavení. Pomocí tohoto objektu se také komunikuje se serverem. Ke komunikaci slouží metoda sendQueryToServer. Metoda odešle na server data ve

formátu DCLData a zakóduje je do formátu JSON. Metoda vždy očekává návratová data s QueryState rovno 100. Pokud je hodnota jiná, došlo k problému a je vygenerován chybový dialog.

4.4 Návrh Wireframe administrace

Když jsem měl představu, jak by měla aplikace fungovat, navrhnul jsem za pomoci drátěných modelů v programu Pencil rozvržení administrace včetně editačního okna. Jednotlivé drátěné modely jsou přiloženy jako příloha této práce ve formátu png na přiloženém CD. Drátěné modely mi usnadnily velkou část mé práce. Přes drobné změny, které jsem si uvědomil až při programování aplikace, nebylo zapotřebí v průběhu mé práce výrazně měnit navržený design programu. Jedinou větší změnou při programování navržené aplikace bylo vyloučení nástroje změna velikosti, u které mi došlo, že v mé aplikaci nedávala smysl. Drátěné modely mi také umožnili vidět hrubý pohled na aplikaci ještě před samotným programováním.

5 Závěr

Během psaní této bakalářské práce jsem prostudoval dostupné informace o nové a perspektivní technologii HTML5 DND API. Kromě tohoto API jsem musel nastudovat i další technologie, které přímo souvisejí s problematikou fotogalerie jako např. práci s objektem canvas, nebo FILE Reader API. Vytvořil jsem sadu příkladů, na kterých jsem testoval podporu technologie mezi aktuálními prohlížeči. Během tohoto testování jsem zjistil velice znatelné rozdíly ve výkonu při zpracování skriptů mezi jednotlivými prohlížeči. Proto jsem nad rámec zadání mé práce provedl měření, které předchozí zjištění potvrdilo. Nečekaně se ukázalo, že ačkoli je Internet Explorer 10 nejnovějším prohlížečem od firmy Microsoft, je ve zpracování JavaScriptu nejpomalejší a to i oproti předchozí verzi Internet Explorer 9. Dále má práce potvrdila můj osobní předpoklad, že je čisté HTML5 DND API ještě velice mladou technologií a pro ostré nasazení se zatím příliš nehodí. Naopak se potvrdilo, že při použití doplňkových knihoven (jako např. jQuery), se technika DND již dá do aplikace s čistým svědomím doporučit. Osobně si ovšem myslím, že by měla být použita pouze jako doplňkový způsob ovládání. Programátor by měl umožnit uživateli i standardní způsob ovládání. Jako důvod nevidím ani tak podporu technologie samotné, ale spíš nezvyk uživatelů na tento druh ovládání. Méně zkušený uživatel by toto ovládání nemusel čekat a mohl by být zmatený.

Pokud bych měl napsat, co mě na této práci nejvíce překvapilo, tak celé programování v jazyce JavaScript. Lépe řečeno asynchronnímu způsobu programování, kdy se při psaní programového kódu musí počítat s mnohdy i více než 5 vlákny souběžně. Docela mě tento paralelní způsob programování oslovil a budu se mu i nadále věnovat. Nejzajímavější mi na jazyku JavaScript přišel systém callbacků, kdy se do parametrů metod vkládají anonymní funkce, které poté "probublávají" kódem zpět.

V rámci mé práce jsem navrhl a zrealizoval univerzální způsob pro datovou komunikaci mezi klientskou aplikací a serverem. Tato abstraktní vrstva by měla ulehčit programátorovi práci, a pokud jí opravdu ulehčí, tak jsem s touto prací spokojen. I po dopsání této práce mám v plánu aplikaci dále vyvíjet a použít ji v mém komerčním projektu.

Seznam použité literatury a zdrojů

[1] W3.ORG. Drag and Drop API [online]. 25.5.2011 [cit. 2012-03-25]. Dostupné z: http://www.w3.org/TR/html5/dnd.html#dnd

[2] W3SCHOOLS.COM. Drag and Drop [online]. 25.3.2012 [cit. 2012-03-25]. Dostupné z: http://www.w3schools.com/html5/html5_draganddrop.asp

[3] HTML5DEMOS.COM. Drag and Drop Demos: automatic upload [online]. 25.3.2012 [cit. 2012-03-25]. Dostupné z: http://html5demos.com/dnd-upload

[4] JOUANNEAU, Laurent. Drag and drop demo in a HTML document: using the HTML5 drag and drop API [online]. 25.3.2012 [cit. 2012-03-25]. Dostupné z: http://ljouanneau.com/lab/html5/demodragdrop.html

[5] ZACHAŘ, Jiří. JavaScript: plynulá změna velikosti obrázku [online]. 15.3.2008 [cit. 2012-03-25]. Dostupné z: http://www.zaachi.com/cs/items/javascript-plynula-zmena-velikosti-obrazku.html

[6] MOZILLA.ORG. How to develop a HTML5 Image Uploader [online]. 5.1.2011 [cit. 2012-03-25]. Dostupné z: http://hacks.mozilla.org/2011/01/how-to-develop-a-html5-image-uploader/

[7] STACKOVERFLOW.COM. Resizing an image in an HTML5 canvas [online]. 23.2.2010 [cit. 2012-03-25]. Dostupné z: http://stackoverflow.com/questions/2303690/resizing-an-image-in-an-html5-canvas

[8] ORGOŇ, Vojtěch. ZDROJAK.CZ. Upload obrázků pomocí HTML5 [online]. 2.2.2011 [cit. 2012-03-25]. Dostupné z: http://zdrojak.root.cz/clanky/upload-obrazku-pomoci-html5/

[9] Selectors Level 4. *W3.org* [online]. 2012 [cit. 2012-12-07]. Dostupné z: http://dev.w3.org/csswg/selectors4/#complex

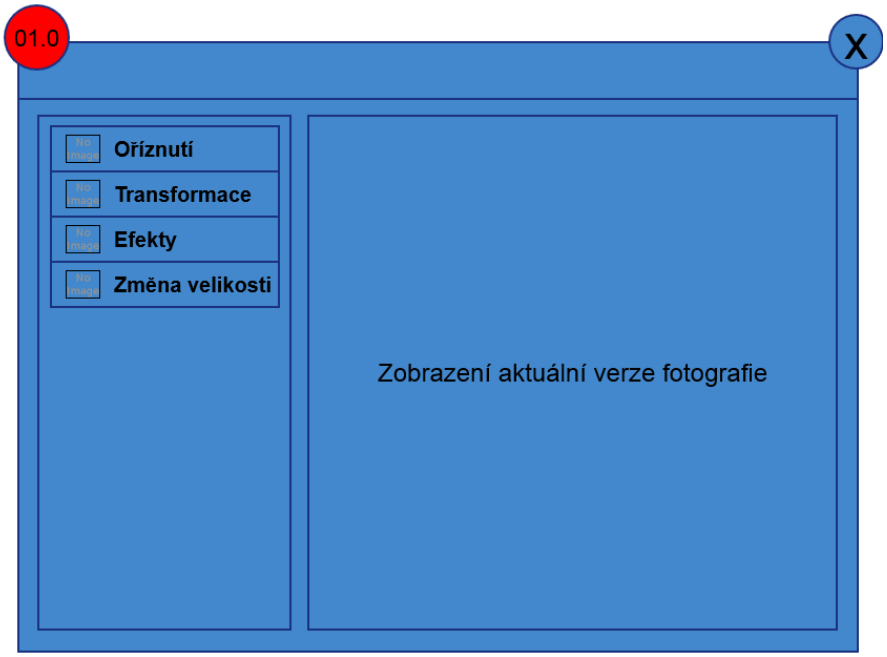
[10] NICHOLAS Z. ZAKAS. *JavaScript pro webové vývojáře: Programujeme profesionálně*. 2009. vyd. Holandská 8, 639 00 Brno: Computer Press, a.s., 2009. ISBN 978-80-251-2509-0.

[11] MALÝ, Martin. Web Sockets. *Zdrojak.cz* [online]. 2009 [cit. 2013-03-03]. Dostupné z: http://www.zdrojak.cz/clanky/web-sockets/

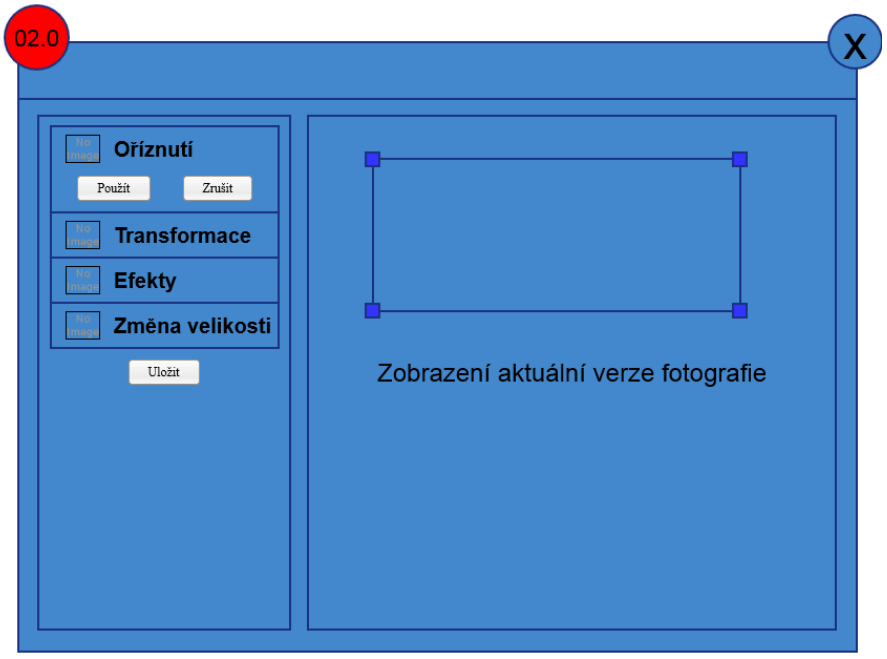
[12] AJAX Introduction. *W3schools.com* [online]. 2011 [cit. 2013-03-03]. Dostupné z: http://www.w3schools.com/ajax/ajax_intro.asp

[13] *Json.org* [online]. 1999 [cit. 2013-03-08]. Dostupné z: http://www.json.org/

Přílohy:



Obr. 8 Wireframe editboxu

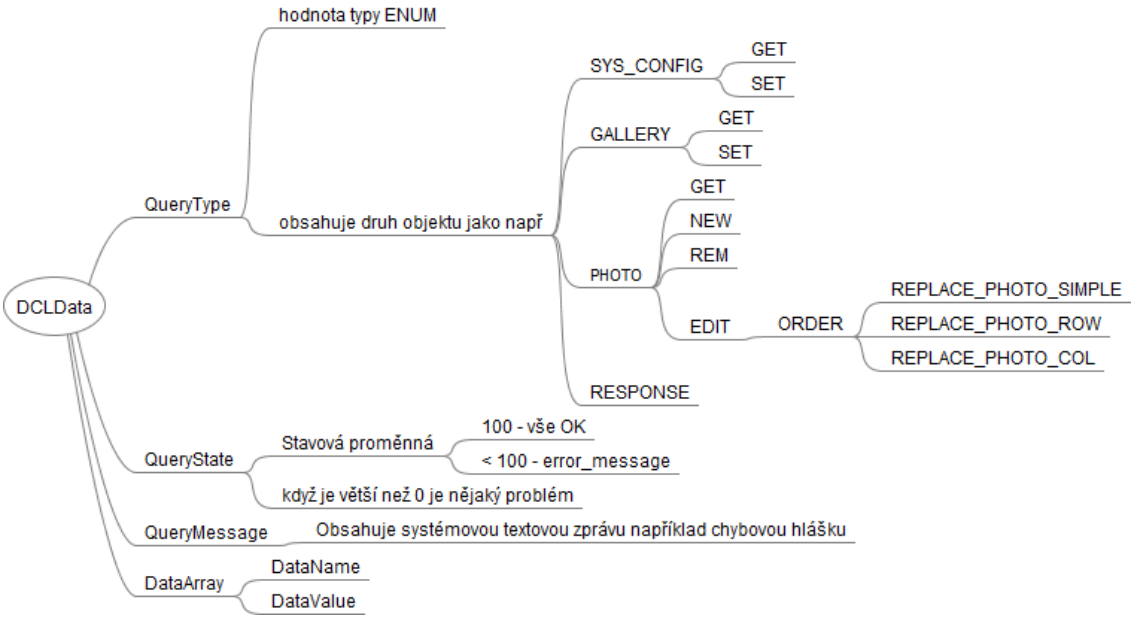


Obr. 9 Nástroj oříznutí

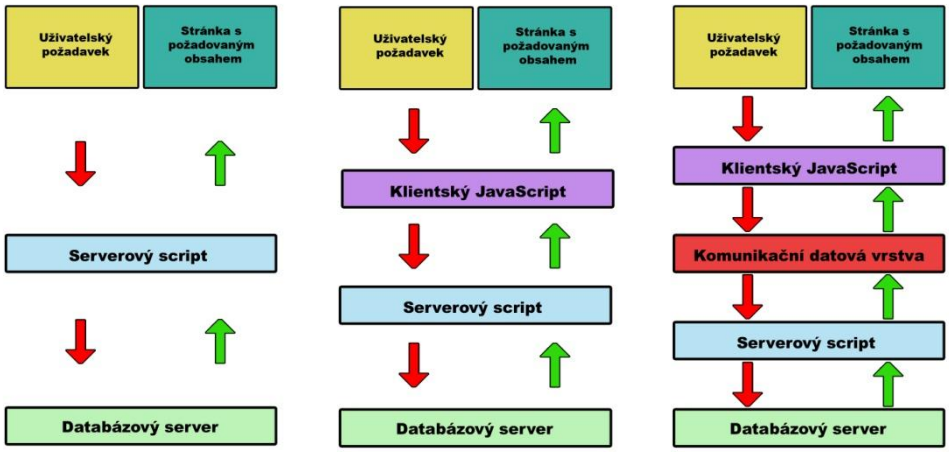
Obr. 10 Nástroj transformace

Obr. 11 Nástroj efekty

Specifikace DCL datového objektu



Aplikační modely



Vlevo: běžná aplikace starého typu

Uprostřed: aplikace moderního typu

Vpravo: Mnou navržený model aplikace využívající abstraktní DCL vrstvu

Tabulky dotazových příkazů

Dotaz na server	Očekávaná odpověď
QUERY_TYPE = SYS_CONFIG_GET DataArray: ORDER = all	QUERY_TYPE = RESPONSE DataArray: PHOTO_THUMB_MAX_WIDTH PHOTO_THUMB_MAX_HEIGHT PHOTO_MAX_WIDTH PHOTO_UPLOAD_FOLDER
QUERY_TYPE = PHOTO_GET DataArray: ORDER = all	QUERY_TYPE = RESPONSE DataArray: gallery_num_row gallery_num_col [id_photo] databázové informace
QUERY_TYPE = PHOTO_NEW DataArray: ORDER = addOriginal	QUERY_TYPE = RESPONSE DataArray: [id_photo] databázové informace
QUERY_TYPE = GALLERY_GET DataArray: ORDER = list	QUERY_TYPE = RESPONSE DataArray: [id_gallery] databázové informace
QUERY_TYPE = PHOTO_REM DataArray: ORDER = photo	QUERY_TYPE = RESPONSE DataArray: removed_photos = id_photo
QUERY_TYPE = PHOTO_EDIT DataArray: ORDER = REPLACE_PHOTO_SIMPLE	QUERY_TYPE = RESPONSE DataArray: ['targetEl']['position_x'] ['targetEl']['position_y']
QUERY_TYPE = PHOTO_EDIT DataArray: ORDER = REPLACE_PHOTO_ROW	QUERY_TYPE = RESPONSE DataArray: [][new_positions]
QUERY_TYPE = PHOTO_EDIT DataArray: ORDER = REPLACE_PHOTO_COL	QUERY_TYPE = RESPONSE DataArray: [][new_positions]

QUERY_TYPE = PHOTO_EDIT DataArray: ORDER = editImg	QUERY_TYPE = RESPONSE DataArray: ['photo']['thumb_file'] ['photo']['thumb_file_name'] ['photo']['big_file']

Systémové hlášky QueryState

Hodnota	Význam
100	Vše proběhlo v pořádku
1	Neoprávněný přístup
2	V databázi nejsou uloženy žádné fotogalerie
3	Nepopsaná chyba
4	Chyba databáze
5	V databázi nejsou uloženy žádné fotogalerie
6	Byla zjištěna narušená integrita dat v databázi
7	Fotografie nemohla být smazána. Již se v databázi nevyskytuje
8	Originál fotografie se nepodařilo uložit na disk serveru