Masarykova Univerzita
Fakulta informatiky

Knižnica pre efektívne nahrávanie obrázkov na webový server

Bakalárska práca

Tomáš Bončo

Brno, Jar 2016

Namiesto tejto stránky vložte kópiu oficiálneho podpísaného zadania práce a prehlásenie autora školského diela.

Prehlásenie

Prehlasujem, že táto bakalárska práca je mojím pôvodným autorským dielom, ktoré som vypracoval samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používal alebo z nich čerpal, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

Tomáš Bončo

Vedúci práce: RNDr. Tomáš Obšívač

Poďakovanie

Chcel by som poďakovať vedúcemu svojej práce za ochotu viesť prácu, priateľský prístup, cenné rady a pomoc pri písaní práce. Ďalej by som chcel poďakovať svojim rodičom za lásku, podporu, starostlivosť a zabezpečenie v každom smere. V neposlednom rade chcem poďakovať svojmu bratovi za lásku a podporu.

Zhrnutie

V práci predstavujeme riešenie nahrávania obrázkov na webový server založené na spracovaní obrázka v prehliadači. Najskôr ho porovnávame so serverovým spracovaním obrázka, kde poukazujeme na výhody spracovania u klienta. Neskôr popisujeme použité technológie, predovšetkým webové komponenty. Potom čitateľa oboznamujeme s naším riešením a možnosťami, ktoré prináša, pričom detailne vysvetľujeme, ako funguje. Na záver naše riešenie porovnáme s inými riešaniami, ktoré tiež spracuvávajú obrázky v prehliadači.

Kľúčové slová

nahrávanie obrázkov, webový server, javascript, webová komponenta, canvas

Obsah

Zoznam tabuliek

Zoznam obrázkov

1 Úvod

Nahrávanie obrázkov na internet je neoddeliteľnou súčasťou našich životov. V roku 1997 konzorcium W3C vydalo štandard HTML 3.2 [4], v ktorom definuje spôsob nahrávania súborov (a teda aj obrázkov) pomocou nepárového tágu HTML <input type="file">. Tento spôsob nahrávania obrázkov pretrváva dodnes, avšak požiadavky sa zmenili. Snaha vývojárov zobrazovať nahraný obrázok pred odoslaním formulára vyžadovala využívanie rôznych trikov (napríklad vkladanie neviditeľných rámcov) alebo iných technológií (Flash, Silverlight). Spomínané spôsoby väčšinou fungujú na princípe, že sa na pozadí obrázok nahraje a následne sa pošle naspäť na stránku, kde sa zobrazí. HTML5 prináša nové možnosti, ako nahrávať obrázky na server.

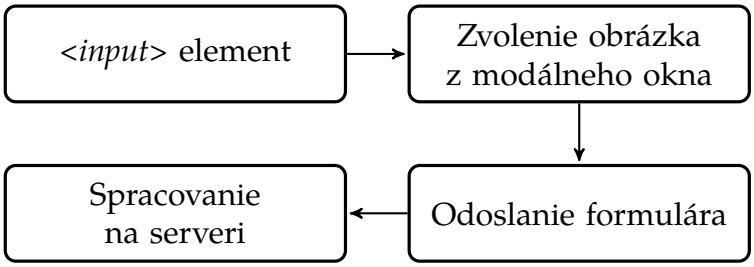
Cieľom tejto práce je vytvoriť knižnicu v jazyku JavaScript, ktorá by zobrazila obrázok pred jeho odoslaním na server. S týmto obrázokom by byť možné posúvať, a nastaviť tak jeho orezanie, a na server sa bude odoslať už iba jeho zmenšená a orezaná časť. Podľa zadania bude takáto knižnica podporovať mobilné zariadenia a tiež bude podporovať nahrávanie viacerých obrázkov. Súčasťou riešenia bude tiež dokumentácia, ktorá vysvetlí, ako knižnicu správne používať. Na záver bude riešenie porovnané s už existujúcimi riešeniami podobného zamerania.

2 Súčasné metódy nahrávania obrázkov

Aby sa dali veci robiť lepšie a efektívnejšie, je najprv potrebné preskúmať už vytvorené spôsoby a uvedomiť si ich nedostatky. Zamedzí sa tak opakovaniu chýb.

2.1 HTML tág <input>

Historicky prvým spôsobom je nepárový tág *<input type="file">*. Objavil sa už v roku 1997, keď konzorcium W3C vydalo štandard *HTML 3.2* [4]. Umožňuje nahrávanie súborov, a teda aj obrázkov, ako súčasť zaslaného formulára (musí sa vyskytovať uprostred tágu <form>). Scénar nahrávania prebieha nasledovne:



Obr. 2.1: Schéma nahrávania obrázka pomocou *<input>* elementu.

1. Užívateľ myšou klikne na *<input>* element.
2. Vyskočí modálne okno, z ktorého užívateľ vyberie obrázok.
3. Užívateľ odošle formulár, obrázok je odoslaný na server, kde je spracovaný.

Výhody tohto riešenia sú predovšetkým:

- Natívna podpora v prehliadačoch - nie sú potrebné žiadne ďalšie JavaScriptové súbory.
- Jednoduché na implementáciu.
- Podpora *Drag&Drop*.

2.2 FormData a XMLHttpRequest

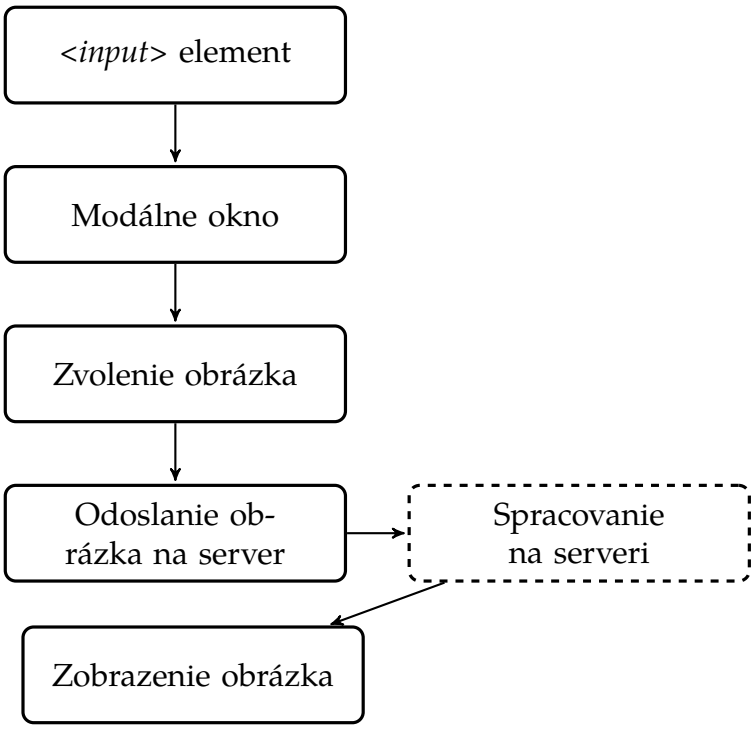
XMLHttpRequest Level 2 pridáva nové rozhranie *FormData*, ktoré umožňuje vytvoriť reprezentáciu formulára v tvare kľúč-hodnota, čím umožňuje zaslať formulár pomocou *XMLHttpRequest* [5]. *XMLHttpRequest* je API, ktoré umožňuje prenášať dáta medzi prehliadačom a serverom bez toho, aby nastalo obnovenie stránky [6].

Vďaka *FormData* je možné odstrániť obmedzenie, kde <*input type="file"*> musí byť vložený do formulára (<form>). Použitím *XMLHttpRequest* eventu *onprogress* je možné sledovať proces nahrávania [7]. Scénar nahrávania môže vyzerať nasledovne:

1. Užívateľ myšou klikne na <*input*> element.
2. Vyskočí modálne okno, z ktorého užívateľ vyberie obrázok.
3. Obrázok sa odošle na server, zatiaľ čo užívateľ sleduje proces nahrávania. Keď je obrázok plne nahraný a spracovaný, odošle sa jeho odkaz naspäť užívateľovi do prehliadača.
4. Užívateľovi sa zobrazí nahraný obrázok. Pokiaľ s ním nie je spokojný, postupuje od bodu 1. Ak je spokojný, odošle formulár. V tomto prípade nedochádza k žiadnej manipulácii s obrázkom.

Výhody tohto riešenia sú predovšetkým:

- Nie je potrebné umiestnenie vo formulári.
- Je možné zaslať obrázok na pozadí, čo umožňuje využitie v moderných webových aplikáciách, kde je obnovenie stránky nežiaduce.
- Počas nahrávania je možné zobraziť proces v percentách.
- Po nahratí obrázka na server je možné obrázok užívateľovi zobraziť. Takto ho užívateľ vidí pred potvrdením formulára.
- Podpora *Drag&Drop*.



Obr. 2.2: Schéma nahrávania obrázka pomocou <*input*> elementu v kombinácií s *FormData* a XMLHttpRequest.

2.3 Riešenia založené na Adobe Flash a Microsoft Silverlight

V dobe písania tejto práce boli stále veľmi rozšírené riešenia, založené na technológii Adobe Flash (ďalej spomínaný už len ako Flash). Flash, pôvodne navrhnutý pre tvorbu multimediálneho obsahu - vektorovej grafiky, animácií, hier pre prehliadač - sa stal populárnym aj na tvorbu väčších aplikácií v prehliadači, pracovanie so vstupm z webovej kamery či mikrofónu, až po prehrávanie videí a zvuku.

O niečo podobné sa pokúšal Microsoft s technológiou Silverlight (ďalej len Silverlight). Obe technológie nepoužívajú Javascript, ale iné jazyky - v prípade technológie Flash ide o ActionScript a v prípade Silverlight ide o Visual Basic, C#, Ruby alebo Python. **V týchto technológiách by preto bolo možné napísať aj celú našu knižnicu**, avšak

nemohli by sme dosiahnuť podporu mobilných zariadení a obe technológie vyžadujú inštaláciu rozšírení tretích strán do prehliadača.

Hlavným dôvodom neúspechu technológie Flash a Silverlight sa stalo odmietnutie podpory od firmy Apple (pozri 3.1). V súčasnej dobe nie je podporovaný na mobilných zariadeniach s operačným systémom iOS, Android a Windows Phone. Samotné Adobe už prestáva ďalej Flash podporovať a zameriava sa radšej na HTML5[8].

3 Nevýhody súčasných metód

3.1 Podpora mobilných zariadení

V roku 2010, Steve Jobs, spoluzakladateľ a v tom čase aj výkonný riaditeľ Applu vydal vyhlásenie *Thoughts on Flash*[9], v ktorom vysvetľuje, prečo zariadenia iPhone, iPod a iPad nepodporujú *Adobe Flash*. Kritizuje *Adobe Flash* z uzavretosti, vysokej systémovej záťaže, slabej bezpečnosti a chýbajúcej podpore dotykových zariadení. BBC v roku 2012 informovala[10], že Adobe sťahuje *Adobe Flash Player* z *Google Play Store*, čo dovtedy umožňovalo prehrávať *Adobe Flash* na mobilných zariadeniach. To znamená, že mobilné operačné systémy, ktoré dohromady zaberajú 96,7% trhu[11] nepodporujú *Adobe Flash*.

3.2 Zbytočné súbory na serveri

Nahrávanie obrázkov bez toho, aby užívateľ najskôr obrázok videl, môže viesť k tomu, že užívateľ nahraje nesprávny obrázok. Tak isto môže vzniknúť problém, kde užívateľ kvôli nesprávnemu spracovaniu obrázka na serveri (napríklad nesprávny orez, pozri 3.3), môže zmeniť svoje rozhodnutie. Takto vznikajú na serveri súbory - obrázky, ktoré sa nikdy nepoužijú. Jedno z riešení tohto problému je pravidelné mazanie nevyužívaných obrázkov.

3.3 Orezanie obrázka

Pri spracovaní na serveri zvyčajne dochádza k zmenšeniu a orezaniu obrázka. Spravidla sa obrázky orezávajú na stred. To však môže byť nesprávne, ako ukazuje obrázok 3.1, kde orezanie na stred odreže osobu - najdôležitejšiu časť obrázka. Čiastočným riešením je využitie detekcie tvárí. Pokročilé riešenia si však vyžadujú pokročilú analýzu obrazu, kde sa analyzuje kontrast a hrany. Vďaka tomu je možné detekovať východ slnka, alebo budovu na obrázku.

Obr. 3.1: Demonštrácia zlého automatického orezania obrázka. Pôvodný obrázok z [3].

3.4 Prenášanie zbytočných dát

Spracovanie obrázka obvykle prebieha na serveri. To však znamená, že ak je žiadaný len zmenšený a orezaný obrázok, tak väčšina dát, ktoré užívateľ preniesol, je zbytočná. Pokiaľ by zmenšenie a orezanie obrázka prebiehalo v prehliadači, tak sa prenesie menej dát, čo zrýchli nahrávanie obrázka na webový server a tiež ušetrí peniaze v prípade spoplatnených mobilných dát.

4 Vytvorenie knižnice na nahrávanie obrázkov

4.1 Ciele

Ciele, ktoré sme stanovili pri vytváraní knižnice majú za úlohu maximalizovať pohodlie užívateľa, minimalizovať zaťaženie pre tvorcov stránky a mala by byť prístupná pre čo najširšie publikum. Preto sme stanovili, že knižnica by mala:

- byť nezávislá od knižníc/frameworkov tretích strán, napr. *jQuery*,
- podporovať nahrávanie, či už zvolením obrázka cez modálne okno alebo aj technológiou *Drag&Drop*,
- byť jednuduchá na implementovanie,
- byť dobre rozšíriteľná o nové funkcie,
- podporovať nahrávanie bez obnovenia stránky,
- podporovať mobilné zariadenia,
- podporovať nahrávanie viacerých obrázkov,
- byť dobre zdokumentovaná.

4.2 Riešenie

4.2.1 Predstavenie

Najskôr sme plánovali využiť *jQuery*, ale kvôli splneniu prvého cieľa (byť nezávislí od knižníc/frameworkov tretích strán) sme sa rozhodli využiť *Custom Element*. Vytvorili sme jeden, ktorý slúži na úpravu obrázka <*x-cupe*> a jeden na podporu nahrávania viacerých obrázkov - <*x-cupe-gallery*>. Oba vychádzajú z rovnakého HTML, pričom <*x-cupe-gallery*> interne využíva <*x-cupe*>, preto sa ďalej v tejto kapitole

zameriame na element <*x-cupe*>. Nahrávanie viacerých obrázkov je bližšie popísané v kapitole 3.5.

Scénar nahrávania môže vyzerať takto:

1. Užívateľ buď pomocou *Drag&Drop* nahraje obrázok, alebo klikne na element a následne si z modálneho okna zvolí obrázok.

2. Obrázok sa zmenší, oreže a zobrazí.

3. Pokiaľ užívateľ nie je spokojný s obrázokom, pokračuje prvým bodom.

4. Pokiaľ užívateľ nie je spokojný s orezaním, držaním stlačeného ľavého tlačidla myši posúva obrázok, dokým nie je spokojný.

5. Keď je užívateľ spokojný s obrázkom aj jeho orezaním, odošle formulár. Až vtedy sa zmenšená a orezaná verzia nahraje na server.

Hoc sa v scenári spomína klikanie, knižnica podporuje aj dotykové zariadenia, a teda všetky úkony je možné vykonať dotykom. Interne oba elementy využívajú nasledujúce HTML:

```
<canvas></canvas>
<input type="file" style="display: none">
<input type="text" style="display: none">
```

Zdrojový kód 4.1: Interná štruktúra elementov <*x-cupe*> a <*x-cupe-gallery*>.

Element <*canvas*> sa stará o zobrazenie obrázka. Na prácu s ním sme vytvorili pomocnú triedu *XCupeCanvasElement*, pomocou ktorej vieme rýchlo zmeniť rozmery. Keď užívateľ klikne na <*x-cupe*>, element deleguje klik na <*input type="file"*>, čo spôsobí, že sa otvorí modálne okno a užívateľ si môže zvoliť obrázok. Pre zjednodušenie práce s ním sme vytvorili triedu *XCupeInputFileElement*. Do <*input type="text"*> sa vloží textová reprezentácia obrázka, čo umožňuje jeho odoslanie cez POST. Opäť, aj pre tento element sme vytvorili pomocnú triedu *XCupeInputFileElement*. Detailné fungovanie knižnice je popísané v sekcii 4.4.

4.2.2 Použitie

Použitie *Custom Elementu* nám umožnuje skutočne jednoduchý spôsob inštalácie, kde sa najskôr načítajú zdrojové súbory knižnice:

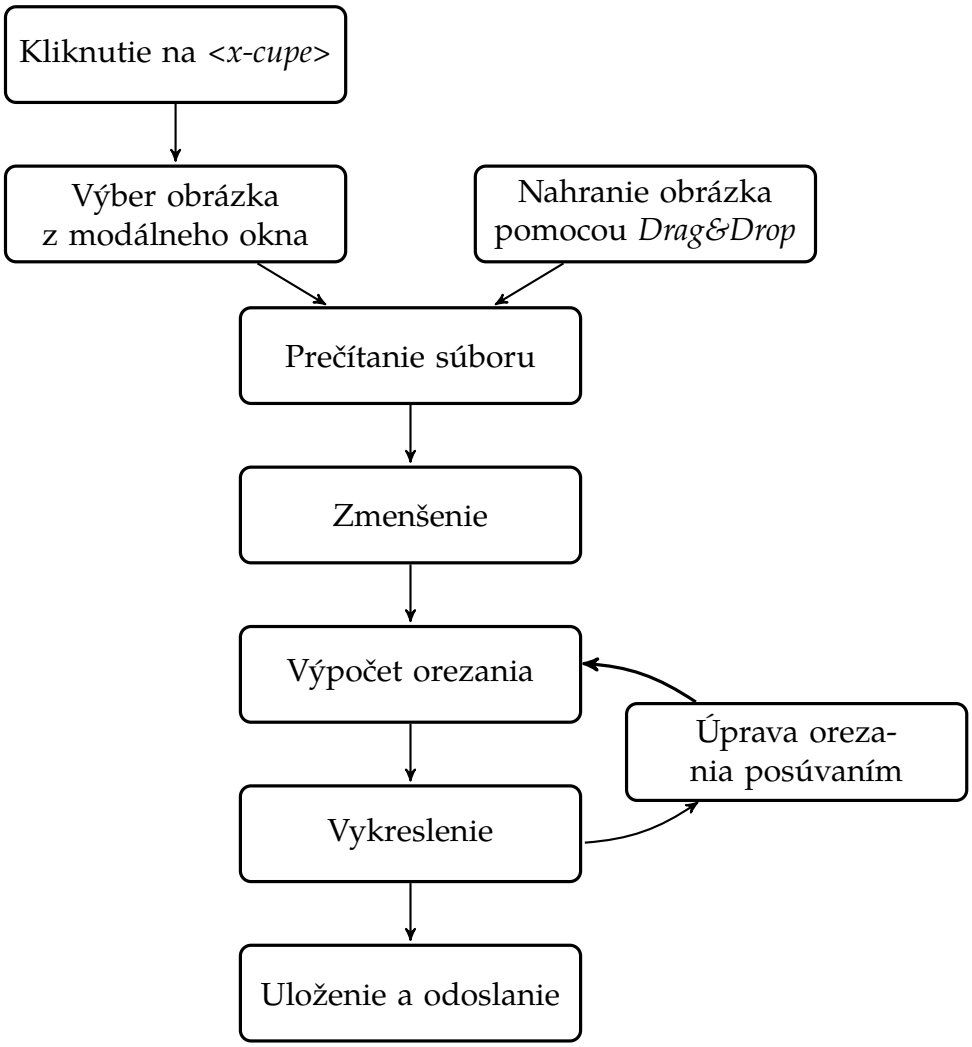
```
<link rel="import" href="dist/x-cupe.html">
```

Následne stačí už len na príslušnom mieste v kóde vytvoriť element:

```
<x-cupe></x-cupe>
```

V prípade nahrávania viacerých obrázkov:

```
<x-cupe-gallery></x-cupe-gallery>
```



Obr. 4.1: Schéma nahrávania obrázka pomocou <*x-cupe*> elementu.

4.3 Použité technológie, služby a postupy

4.3.1 Webové komponenty

Webové komponenty (v originále Web Components) pozostávajú zo štyroch oddelených technológií - *Custom Elements*, *HTML Templates*, *Shadow DOM* a *HTML Imports*. Využívajú sa na vytvorenie znovu použiteľných komponentov užívateľského rozhrania. Keďže sú [1] súčasťou prehliadačov, nepotrebujú externé knižnice ako *jQuery* alebo *Dojo*. Existujúce webové komponenty je možné používať jednoduchým importovaním v HTML, teda bez písania dodatočného (JavaScriptového) kódu. [12]

V dobe písania tejto práce boli webové komponenty podporované plne len v *Chrome* a v pokročilom stave vývoja v prehliadači *Firefox*[2]. Podrobnejší popis podpory uvádzame v tabuľkách 4.1 a 4.2[3]. Vzhľadom k tomuto stavu v práci používame náhradu (tzv. „polyfill“) *webcomponents.js* [14]. Tá poskytuje podporu pre webové komponenty aj do všetkých aktuálnych dnes používaných prehliadačov, ktoré ich ešte nemajú zabudované. Výnimku tvorí *Internet Explorer 10*, ktorý nepodporuje *Custom Elements* a *HTML Imports*. [14]

Custom Elements umožňujú vytváranie vlastných HTML elementov. Tie môžu mať svoje vlastné CSS štýly a správanie. Kľúčovou súčasťou sú udalosti životného cyklu, ktoré umožňujú definovať sprá-

1. Ako vysvetľuje ďalší odstavec, podpora v čase písania tejto práce nebola úplná.
2. Funkciu je možné zapnúť v *about:config*
3. V tabuľke neuvádzame prehliadač *Opera*, pretože využíva rovnaké vykresľovacie jadro ako *Chrome* [13]

	IE 11	Edge	Chrome	Safari	Firefox
Templates	Nie	13+	26+	7.1+	22+
HTML Imports	Nie	Nie	47+	Nie	Nie[2]
Custom Elements	Nie	Nie	47+	Nie	Nie[2]
Shadow DOM	Nie	Nie	47+	Nie	Nie[2]

Tabuľka 4.1: Podpora webových komponentov v prehliadačoch pre počítače [1]. [2]

	Android Browser	iOS Safari	Opera Mini
Templates	4.4+	8.4+	Nie
HTML Imports	47+	Nie	Nie
Custom Elements	4.4.4+	Nie	Nie
Shadow DOM	4.4+	Nie	Nie

Tabuľka 4.2: Podpora webových komponentov v mobilných prehliadačoch [1]. [2]

vanie v prípade, že bol element vytvorený, vložený do *DOM*[4], odstránený z *DOM* alebo sa zmenili atribúty. [12]

HTML Templates je v podstate označenie pre element <*template*>, mechanizmus, ktorý umožňuje uložiť obsah v prehliadači bez toho, aby ho vykresľoval [1]. Pomocou JavaScriptu je však jeho obsah možné čítať a ďalej s ním pracovať.

Shadow DOM poskytuje zapuzdrenie *HTML*, *CSS* a *JavaScript*u do webového komponentu. Robí to tak, že tieto nebudú súčasťou *DOM* alebo hlavného dokumentu. Hlavná výhoda spočíva predovšetkým v štýlovaní, kde použitie *Shadow DOM* zaručuje, že *CSS* iných časti stránky neovplyvňujú vzhľad komponentny, a komponent nemôže ovplyvniť vzhľad iných častí stránky. [15]

4.3.2 Canvas

Canvas je nový tág *HTML5*, ktorý umožňuje pomocou JavaScriptu vykresľovať grafy, upravovať fotografie, animovať a dokonca aj spracovať videá. Využíva ho aj *WebGL* na hardvérovo akcelerovanú 3D grafiku na webe [16].

V našej práci ho využívame na zobrazovanie obrázka. Samotná knižnica už obsahuje potrebné nástroje na vkladanie a zmenšovanie obrázka, čo nám uľahčuje prácu a zároveň nám <*canvas*> umožňuje prístup k jednotlivým pixelom, čo ocenia najmä tvorcovia pokročilých rozšírení (napr. detekovať nevyužívanú plochu obrázka alebo prevod do čierno-bielej verzie). Keď užívateľ presúva obrázok kvôli

4. Document Object Model

orezaniu, prekresľujeme daný výrez do <*canvas*> pomocou *requestAnimationFrame*[17] pre maximálnu plynulosť.

4.3.3 Drag&Drop

Drag&Drop je novinkou *HTML5*. Umožňuje predovšetkým presúvanie elementov na stránke, čím rieši problémy, ktoré sa zložito snažili vyriešiť knižnice tretích strán (napr. *jQuery*). Okrem toho však umožňuje aj presúvanie súborov z počítača do okna prehliadača, z čoho ťaží aj táto práca.

4.3.4 File Reader

File Reader umožňuje čítať obsah súborov vo webovom prostredí. Tie môžu byť načítané buď pomocou tágu <*input*> alebo pomocou *Drag&Drop* (pozri 4.3.3)[18][5].

Použili sme ho práve pre prečítanie zvolených súborov, ktoré sme následne zobrazili v <*canvas*> (pozri 4.3.2).

4.3.5 Promise

Promise je objekt určený pre asynchrónne výpočty a výpočty naplánované na neskôr (napríklad cez *setTimeout* alebo *setInterval*). Nevracia priamo metódu, ale vracia „prísľub“, že niekedy v budúcnosti hodnota bude dostupná [19]. V silno asynchrónnej aplikácii zabraňuje tzv. „callback hell“, kde množstvo callbackov bráni zrozumiteľnosti a ladeniu kódu.

Promise používame naiesto predávania callbackov, predovšetkým pri čítaní užívateľovho obrázka.

4.3.6 TypeScript

TypeScript je typová nadstavba JavaScriptu, ktorá sa prekladá do JavaScriptu. Tým, že sa rýchlo vyvíja, okrem typovosti prináša aj funkcie *ECMAScript 2015 (ES6)*, čo umožňuje programátorom používať

5. Citovaný MDN udáva ešte aj HTMLCanvasElement.mozGetAsFile(), ktorý je neštandardizovaný a v dobe písanie tejto práce podporovaný iba prehliadačom Firefox.

funkcie, ktoré v čase písania tejto bakalárskej práce ešte nie sú implementované v prehliadačoch.

Naša knižnica využíva TypeScript, pretože umožňuje rýchlejšie odhaľovať chyby, vedie k písaniu rozhraní, čo môže značne pomôcť tvorcom rozšírení, a tiež kvôli podpore *ECMAScript 2015*. Navyše, jeho vlastnosť, že sa prekladá do JavaScriptu nijak neobmedzuje tvorcov rozšírení alebo jeho celkové použitie.

4.3.7 Mocha

Mocha je JavaScriptový framework používaný na jednotkové testy. Známy je predovšetkým pre jednoduchý zápis testov a prehľadnú dokumentáciu. Zvolili sme ho, pretože autori tejto práce s ním majú predchádzajúce skúsenosti.

4.3.8 GitHub

Počas tvorby sme používali Git a výsledok našej práce sme uverejnili v službe GitHub - webovej službe na správu Git repozitárov. Git je (v čase písania tejto práce) bezplatný nástroj na spravovanie verzií. Na rozdiel od Apache Subversion (SVN) je distribuovaný. GitHub je server pre Git, ktorý však pridáva vlastné funkcie. Pokým Git je nástroj ovládaný príkazovým riadkom, GitHub poskytuje webové rozhranie. Ďalej poskytuje nástroje pre lepšiu spoluprácu, ako wiki stránky a jednoduchú správu úloh, umožňuje nastaviť prístupové práva, a tiež možnosť kopírovať repozitáre medzi užívateľmi [20].

4.4 Postup nahrávania a nastavenia

V tejto sekcii sa budeme venovať samotnému procesu nahrávania obrázka, od jeho zvolenia, úpravy po odoslanie na server.

4.4.1 Zvolenie obrázka

Užívateľ môže obrázok zvoliť dvoma spôsobmi - výberom z modálneho okna alebo pomocou *Drag&Drop*. V prípade nahrávania obrázka

cez modálne okno, užívateľ musí kliknúť na <*x-cupe*> element, do ktorého chce obrázok nahrať. Interne sa tento klik prevedie na <*input type="file"*> a to spôsobí otvorenie modálneho okna. Po tom, čo si užívateľ zvolí obrázok, je tento súbor pomocou *File Reader* (pozri 4.3.4) prečítaný a pripravený na ďalšie spracovanie. V prípade, že chceme tento spôsob nahrávania zakázať, stačí nastaviť atribút *allow-select* na hodnotu *false*.

V prípade použitia *Drag&Drop*, a teda označenie obrázka, jeho presunutie ponad cieľový element a následné pustenie myši/oddialenie prsta od dotykovej plochy, je postup podobný. Element (<x-cupe>) vie pomocou zachytenia udalosti *drop* zachytiť presúvaný obrázok a ďalej ho rovnako ako pri modálnom okne pomocou *File Reader* prečítať. Ak chceme tento spôsob zakázať, je potrebné nastaviť atribút *allow-drop* na hodnotu *false*.

4.4.2 Zmenšenie a orezanie

Po zvolení obrázka spravidla prebieha jeho zmenšenie a orezanie. Pre túto činnosť je potrebné vedieť rozmery nahrávaného obrázka a tiež rozmery výsledného obrázka, pričom sú implementované tieto scenáre:

Nahrávaný obrázok je väčší a mal by sa zmenšiť; orezávanie je zapnuté

Očakávame, že práve tento scénar bude najvyužívanejší. V tomto scenári sa obrázok pomerne zmenší tak, aby pomerne kratšia strana nahrávaného obrázka bola zhodná s príslušnou stranou finálneho obrázka. To je z toho dôvodu, aby užívateľ mohol hýbať obrázkom v ose, ktorá je pomerne dlhšia (a teda v tej, v ktorej má posúvanie zmysel).

Nahrávaný obrázok je väčší a mal by sa zmenšiť; orezávanie je vypnuté

V tomto prípade sa snažíme vložiť celý nahrávaný obrázok do finálneho obrázka. Dosiahneme to pomerným zmenšením obrázka tak, aby pomerne dlhšia strana bola zhodná s príslušnou stranou finálneho obrázka.

Nahrávaný obrázok je menší a mal by si zväčšiť; orezávanie je zapnuté

Cieľom našej knižnice je, aby správca stránky vždy dostal z prehliadača taký obrázok, aký požaduje. Nemusí potom upravovať obrázok na serveri. To však znamená, že môže nastať prípad, že užívateľ zvolí menší nahrávaný obrázok, než je požadovaný finálny obrázok. V tom prípade prebieha spracovanie obrazu rovnako, ako v prvom scenári, len miesto zmenšovania sa obrázok zväčšuje.

Nahrávaný obrázok je menší a mal si zväčšiť; orezávanie je vypnuté

Postupujeme rovnako ako v prípade s väčším obrázkom. Miesto zmenšovania však obrázok zväčšujeme.

Finálny obrázok má len jeden fixný rozmer

Tento scenár je vhodný, ak vytvárame napr. obrázkové galérie, pričom chceme, aby obrázky mali pevnú šírku a ľubovoľne dlhú výrku. Ak chceme variabilnú šírku, je potrebné nastaviť atribút *width* elementu <*x-cupe*> na hodnotu *-1*. V prípade, že chceme variabilnú výšku, je potrebné obdobne nastaviť parameter *height* na hodnotu *-1*. V tomto scenári sa obrázok pomerne zmenší podľa fixnej príslušnej strany finálneho obrázka. Druhá strana finálneho obrázka sa nastaví pomerne k zmenšej výške finálneho obrázka. V tomto scenári nie je možné obrázok orezávať.

Finálny obrázok nemá fixný rozmer

Tento scénar je vhodný, ak chceme aby užívatelia, mohli zaslať obrázok v ľubovoľných rozmeroch a zároveň ho pred odoslaním mohli vidieť. Finálny obrázok bude mať rozmery nahrávaného obrázka a ten teda nie je zmenšovaný ani orezávaný. Pre vytvorenie tejto situácie treba nastaviť aj atribút *width*, aj atribút *height* na hodnotu *-1*.

Obrázok sa orezáva posúvaním obrázka (pozri 4.4.3) a základným nastavením je orezanie na stred. To sa dá zmeniť atribútom *align*, pričom valídne hodnoty pozostávajú z kombinácie slov *left*, *right*, *top*, *bottom* a *center*, napríklad: <*x-cupe align="left bottom"*>. Pri zadaní iba jednej osi pre zarovnanie sa automaticky predpokladá, že druhá sa

má zarovnať na stred. Orezanie sa dá úplne vypnúť, a to nastavením atribútu *crop* na hodnotu *false*. Vnútorne knižnica pracuje tak, že sa textové hodnoty prerátajú na odsadenie obrázka zľava a zhora, ale obrázok sa nijako nemodifikuje. To nastáva až pri vykreslení.

Takisto sme si všimli, že ak zmenšíme veľmi veľký obrázok, jeho kvalita je nízka. Oproti pôvodnému obrázku je zdanlivo ostrejší, a to je spôsobené nevyhovujúcim spôsobom prekresľovania obrázka, tzv. *downsampling*-om. Problém je, že prehliadač musí z plochy napríklad 4x4 pixely vytvoriť iba jeden pixel a aby to spravil najrýchlejšie, jeden z nich vyberie, pričom sa neberie do úvahy kontext. Preto sme aplikovali jednoduchý algoritmus, ktorý obrázok zmenší postupne. Počet prekreslení je možné nastaviť atribútom *quality* a jeho predvolená hodnota je 3. Čím vyššia táto hodnota je, tým dlhšie trvá proces od zvolenia obrázka po jeho vykreslenie. Keď je táto hodnota príliš nízka (napríklad 1), tak je slabšia kvalita. Zmenšený obrázok sa uloží, aby pri jeho ďalšej manipulácií ho nebolo potrebné opäť zmenšovať.

4.4.3 Posúvanie obrázka

Pokiaľ je povolené orezanie obrázka, je možné toto orezanie meniť posúvaním obrázka. Aby sme odlíšili kliknutie, kedy má vyskočiť modálne okno s možnoťou výberu obrázka, a samotné posúvanie, nastavili sme limit, počas ktorého musí užívateľ držať myš stlačenú na 100ms. Rovnaký limit platí aj pre dotykové zariadenia. Po spustení udalosti presúvania sa vyrátava súčasná pozícia myši a porovnáva sa s pozíciou, na ktorej začalo presúvanie. Rozdielom týchto hodnôt získavame informáciu o tom, ako máme posunúť obrázok. Tento rozdiel sa následne priráta k orezaniu obrázka. Nakoniec už zmenšený obrázok s novým nastavením orezania vykreslíme s využitím *requestAnimationFrame*. Posúvanie obrázka sa dá vypnúť nastavím atribútu *allow_move* na hodnotu *false*.

Vzhľadom k tomu, že ukladanie (pozri 4.4.4) je časovo náročná operácia, počas posúvania k nemu nedochádza. Keď ale užívateľ posúvanie ukončí, obrázok je uložený.

4.4.4 Odoslanie obrázka

Pre odoslanie obrázka stačí vložiť element (<*x-cupe*>) do elementu <*form*> a nastaviť mu atribút *name*. Po odoslaní formulára sa obrázky prenášajú vo formáte PNG ako Base64 reťazce. Interne každý <*x-cupe*> element obsahuje <*input type="text"*>, ako je spomenuté v sekcii **??**. Tomu sa nastaví *name* a jeho hodnota je odoslaná na server (po odoslaní formulára). Keďže nevieme, kedy užívateľ odošle formulár, obrázok sa do inputu priebežne ukladá - po načítaní obrázka a potom, ako skončí jeho posúvanie (pozri 4.4.3).

Samotné ukladanie je časovo náročná operácia, ktorá môže byť pri použití *XMLHttpRequest* úplne zbytočná. Preto je možné túto funkciu vypnúť jednoduchým neuvedením atribútu *name* a využívať metódu *getContent()* elementu <*x-cupe*>. Vďaka nej je možné získať prístup priamo k vlastnosti *context* elementu <*canvas*>, alebo uložiť obrázok vo formátoch *PNG*, *JPG* a *WEBP*[6]. V prípade formátov *JPG* a *WEBP* je tiež možné nastaviť kvalitu v rozsahu od 0 (0%) po 1 (100%), predvolená hodnota je 0.92. Obrázky sú ukladané (a odosielané) v 96 dpi [21]. Funkcia vracia obrázok ako Base64 reťazec. Použitie znázorňuje ukážka 4.2.

```
var xcupe = document.querySelector('<x-cupe>');
var content = xcupe.getContent( 1, 'image/jpg', 0.85 );
```

Zdrojový kód 4.2: Použitie metódy getContent() pre získanie obrázka.

V prípade elementu <*x-cupe-gallery*> sa pridáva na začiatok ďalší parameter, ktorý špecifikuje tvar, v akom má vrátiť výsledok, a to buď ako pole („array“) alebo ako mapu („map“). V prípade poľa vracia obrázky z <*x-cupe*> zaradom. V prípade mapy vracia mapu, kde je elementu priradený obsah.

Nahrávanie viacerých obrázkov cez formulár zatiaľ nie je možné. Keďže *Shadow DOM* vytvára HTML štruktúru, ktorá ale nie je súčasťou DOM, formulár (element <*form*>) nemá prístup k elementom <*input*> v Shadow DOM, a teda nie je možné obrázky poslať. Tento problém sa dá vyriešiť dvoma spôsobmi:

- *Custom Element* využíva *Shadow DOM* na internú reprezentáciu, ale tiež môže reflektovať aj obsah, ktorý je vložený dovnútra ele-

6. Nemusí byť podporovaný v každom prehliadači

mentu v DOM na konkrétne miesto v *Shadow DOM*. Problémom však je, že sa tvorcovia prehliadačov nevedia dohodnúť na tom, ako by to malo fungovať [22]. Kým v pôvodnej špecifikácii sa objavuje nový element <*content*>, *MDN*[7] ho už neodporúča používať [23]. Miesto toho navrhuje používať element <*slot*>, lenže jeho presná špecifikácia a podpora nie je známa. Ani náhrada (tzv. „polyfill“) *webcomponents.js*, ktorú používame, aby sme dosiahli maximálnu možnú kompatibilitu s používanými prehliadačmi ho zatiaľ nepodporuje.

- Vložiť element <*input*> do *CustomElement*u, ale nie do *Shadow DOM*. Je to jednoduchý trik. Keďže takýto element je súčasťou DOM, ale nie je súčasťou *ShadowDOM*, prehliadač nevie, ako ho má vykresliť, a preto ho ani nevykreslí. Avšak pri odosielaní formulára jeho obsah odosiela. Tento trik využívame v elemente <*x-cupe*>, avšak nemôžeme ho využiť v <*x-cupe-gallery*>, pretože ak by sme nevkladali elementy <*x-cupe*> v <*x-cupe-gallery*> do *ShadowDOM*, neboli by viditeľné a to je nežiadúce.

4.5 Nahrávanie viacerých obrázkov

Nahrávanie viacerých obrázkov je možné pomocou párového tágu <*x-cupe-gallery*>. Ten interne využíva elementy <*x-cupe*>, ktoré vytvára vždy pri zvolení nového obrázka, či už pomocou modálneho okna alebo pomocou *Drag&Drop*. Element <*x-cupe-gallery*> podporuje všetky atribúty ako <*x-cupe*>, avšak pri ich zmene sa táto zmena pošle všetkým *x-cupe* elementom. Výnimkou sú atribúty *allow-drop* a *allow-select*, ktoré sa aplikujú len pre <*x-cupe-gallery*> a v <*x-cupe*> sú vypnuté.

4.6 Rozšíriteľnosť

Nakoľko naša knižnica používa na spracovanie a zobrazovanie obrazu tág <*canvas*>, využíva a aj poskytuje možnosť pracovať s obrazom na úrovni jednotlivých pixelov. To umožňuje vytváranie pokro-

7. Mozilla Developer Network - najväčšia a najlepšia dokumentácia k webovým technológiám.

čilých rozšírení, ktoré napríklad dokážu zmeniť farebné odtiene, využívať detekciu hrán, rozpoznávanie tvárí a podobne.

4.6.1 Systém rozšírení

Predstavme si dve rozšírenia - „zoom“ (rozšírenie umožňujúce zväčšovať a zmenšovať obraz) a „black&white“ (rozšírenie, ktoré prevedie obraz do čierno-bielej škály). Chceli sme, aby tvorca stránky vedel stanoviť, ktoré <*x-cupe*> elementy budú používať „black&white“, a ktoré nie. Tiež sme chceli, aby vedel nastaviť rozšírenie „zoom“ pre všetky <*x-cupe*> elementy. **Naším cieľom teda bolo, aby rozšírenie bolo možné aplikovať aj pre jednotlivé (<x-cupe>) elementy, ale aj univerzálne - pre všetky (<x-cupe>) elementy**. Naším ďalším cieľom bolo, aby **rozšírenia bolo možné kombinovať**, a teda aby tvorca stránky vedel nastaviť obe rozšírenia - „zoom“ aj „black&white“ na rovnaký <*x-cupe*> element. Naším posledným cieľom bolo, aby boli rozšírenia nezávislé moduly - samostatné súbory, ktoré o sebe nevedia (resp. nemusia vedieť).

Zhodnotili sme existujúce riešenia a rozhodli sme sa, že rozšírenia budú funkcie, ktoré budú upravovať metódy našej knižnice. Tým dosiahneme „cibuľový efekt“.

4.6.2 Cibuľový efekt

Pre lepšie vysvetlenie uvádzame vzorku kódu zo vzorového rozšírenia „zoom“ (pozri 4.6.4), na ktorom následne vysvetlíme „cibuľový efekt“ a jeho výhody.

```
var self = this;
var originalRnDImg = controller.readAndDrawImage; // (A)
controller.readAndDrawImage = function() // (B)
{
    return originalRnDImg.apply( self, arguments ) // (C)
    .then( function()
    {
        // ... (D)
    }
}
```

Zdrojový kód 4.3: Vytváranie cibuľového efektu.

(A) Do premennej *originalRnDImg* uložíme pôvodnú metódu *readAndDrawImage XCupeController* triedy.

(B) Prepíšeme pôvodnú metódu *readAndDrawImage* novou funkciou, ktorá

(C) spustí pôvodnú metódu, a keď tá prebehne úspešne, tak

(D) vykoná doplňujúci kód.

Je potrebné si uvedomiť, že metóda *readAndDrawImage* ostáva zmenená a keď k nej bude pristupovať ďalšie rozšírenie (napr. „black&white“ popísaný v 4.6.1), opäť pridá ďalšiu vrstvu. Následne, keď bude metódu volať *XCupeController*, zavolá modifikáciu z „black&white“, ktorá zavolá modifikáciu z rozšírenia „zoom“, ktoré zavolá pôvodnú metódu.

Tým sme dosiahli, že:

1. Je možné aplikovať súčasne niekoľko rozšírení na jeden <*x-cupe*> element.

2. Jednotlivé rozšírenia o sebe nevedia. Všetky pristupujú (v prípade ukážky) k *XCupeController*.

4.6.3 Aplikovanie rozšírenia

Rozšírenie je funkcia, pričom odporúčame, aby vždy dávala možnosť aplikovať rozšírenie zvlášť na jednotlivé prvky, ale tiež aj na všetky prvky. Vo vzorovom rozšírení „zoom“ (pozri 4.6.4) sme to dosiahli prvým parametrom, ktorý, ak je prázdny, tak sa aplikuje rozšírenie na triedu *XCupeController*. V prípade, že prázdny nie je, predpokladáme, že obsahuje odkaz na už vytvorený element, a teda rozšírenie aplikujeme na ňom. Pre plné porozumenie uvádzame príklad:

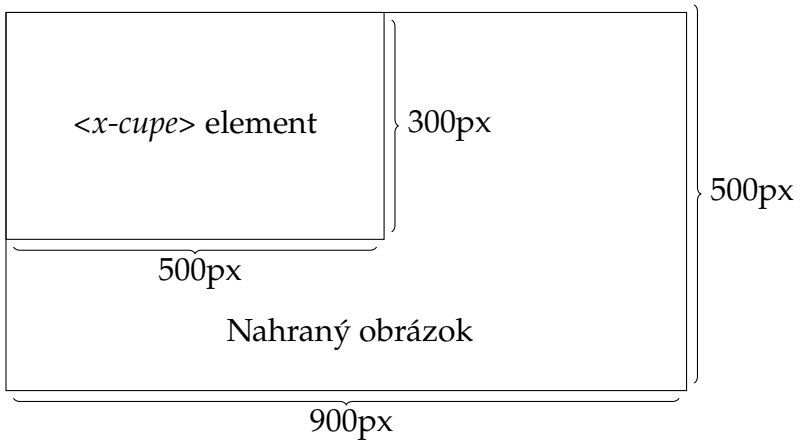
```
// Aplikuje zoom pre~v etky XCupe elementy
XCupeZoom();

// Aplikuje zoom pre~vybran  XCupe element
var cupeElement = new HTMLXCupeElement();
XCupeZoom( cupeElement );
```

4.6.4 Vzorové rozšírenie

Praktická časť tejto práce, v zložke *plugins* obsahuje vzorové rozšírenie *zoom*. Toto rozšírenie má za úlohu umožniť užívateľom zväčšovať a zmenšovať obrázok pred odoslaním.

Keď užívateľ obrázok vloží, vyráta sa pomer medzi zobrazovanou plochou a skutočnou veľkosťou obrázka a vyráta sa maximálne oddialenie. Za najnižšie oddialenie (alebo maximálne priblíženie) je určená hodnota, kde jeden pixel nahrávaného obrázka zodpovedá jednému zobrazenému pixelu. Následné točenie stredným stlačidlom myši mení pomer, akým sa pôvodný obrázok zmenší predtým, než sa vykreslí.



Obr. 4.2: Nákres možného scenára nahrávaného obrázka.

Pre lepšie porozumenie opíšeme postup pre vzorový scénar popísaný obrázkom 4.2.

Inicializácia. Na začiatku sa určí minimálne priblíženie (vtedy vidíme najväčšiu časť obrázka) na hodnotu 1, maximálne priblíženie na hodnotu 1, aktuálne priblíženie na hodnotu 1 a krok na 0,1.

Nahranie obrázka. Po nahraní obrázka sa vyráta pomer šírky (900/500 = 1,8) a výšky (500/300 = 1,6) medzi nahrávaným obrázkom a rozmermi <*x-cupe*> a menší z týchto pomerov sa uloží ako maximálny zoom.

Vykreslenie obrázka. Element <*x-cupe*> sa štandardne snaží zmenšiť obrázok tak, aby bolo vidno čo najviac jeho obsahu pri zachovaní pomerov strán. To znamená, že v tomto prípade nastaví výšku na 300px, a šírku na 540px, pričom vykreslí 500px a umožní užívateľovi posúvať obrázok po šírke o 40px. V rozšírení *zoom* sa na konci tohto výpočtu prenásobia oba rozmery hodnotou aktuálneho priblíženia, čím sa pomerne zväčšia. V prípade priblíženia s hodnotou 1,4 budú teda rozmery 420px a 700px.

Zväčšovanie obrázka. Pri točení stredným tlačidlom myši sa určí smer - +1 pre točenie smerom hore a -1 pre točenie smerom dole a vynásobia sa krokom. Získame tak hodnotu buď 0,1 alebo -0.1 a túto hodnotu prirátame k aktuálnemu priblíženiu.

4.7 Spracovanie obrázka na PHP serveri

Spracovanie obrázka na PHP serveri je veľmi jednoduché. Nakoľko sa na PHP server posiela textový reťazec, obsahujúci hlavičku a samotný obrázok zakódovaný do *Base64* reťazca a zároveň jazyk PHP už obsahuje funkciu *imagecreatefromstring()*, ktorá dokáže vytvoriť obrázok z *Base64* reťazca, vzorové uloženie prebieha v 3 krokoch:

1. Oddelenie hlavičky od obsahu obrázka (jeho *Base64* reprezentácie) cez *explode()*.
2. Vytvorenie obrázka z Base64 reťazca pomocou *imagecreatefromstring()*.
3. Uloženie obrázka funkciou *imagepng()*. Naše vzorové spracovanie v tomto kroku tiež generuje pre obrázok nový názov.

Vzorové riešenie počíta s odoslaním formulára (elementu <*form*>), kde sa obrázky odosielajú vždy vo formáte PNG. V prípade posielania a ukladania obrázkov v iných formátoch je riešenie obdobné.

Nakoľko obrázok je už zmenšený a orezaný, jeho ďalšie spracovanie nie je potrebné.

4.8 Dokumentácia

5 Validácia riešenia

5.1 Podpora mobilných zariadení

Naše riešenie sme testovali na dvoch zariadeniach - LG G2 (Android 5.0.2, natívny prehliadač aj Chrome) a iPhone 6 (iOS 9.3.1, Safari aj Chrome) a v oboch prípadoch boli výsledky úspešné. Cieľom bolo overiť, či naše riešenie funguje správne na dvoch najväčších platformách. Implementácia sa pre mobilné prehliadače jemne líši. Kým užívatelia na stolových počítačoch používajú v prevažnej miere myš, mobilné zariadenia sú dotykové, a teda bolo potrebné zabezpečiť, aby sa dotyky správne interpretovali - používanie *event.changedTouches[0].pageX* miesto *event.pageX* a tiež povolenie posúvania obrázka len jedným prstom.

Takisto sme si všimli, že náhrada („polyfill“) *webcomponents.js*, ktorú používame, aby sme dosiahli plnú podporu na dosiaľ nepodporovaných prehliadačoch, nesvytvára pseudotriedu *:host* na mobilných zariadeniach, preto je potrebné do CSS manuálne vložiť spôsob, akým sa má element vykresliť, ako uvádza ukážka 5.1.

```
x-cupe, x-cupe-gallery { display: inline-block; }
```

Zdrojový kód 5.1: Štýly potrebné pre správne fungovanie na mobilných zariadeniach.

5.2 Porovnanie s ďalšími riešeniami

Vzhľadom k tomu, že knižníc na nahrávanie obrázkov je veľa, rozhodli sme sa porovnať tú našu s vybranými ostatnými. Porovnávali sme spôsob a rýchlosť nahrávania, spôsob spracovania a ponúkané možnosti. Všetky knižnice musia upravovať obrázok v prehliadači. Testy sme vykonávali s dvoma obrázkami - s obrázkom ABSTRACT (3000 × 2000px; 72dpi; 5,38MB) a s väčším obrázkom PANORAMA (24442 × 4195px; 150dpi; 28,1MB).

http://scottcheng.github.io/cropit/ (A) (30. 4. 2016)
je jQuery rozšírením, umožňuje nahrávať obrázky pomocou *Drag&Drop*

aj vybraním obrázka cez modálne okno. Podporuje približovanie (len pomocou posuvnika), a tiež rotovanie obrázka. Na vykresľovanie obrázka nepoužíva <*canvas*>, ale <*img*>, s tým, že keď je potrebné obrázok uložiť, prekreslí vybraný výsek obrázka do elementu *canvas*, z ktorého výsledok vyexportuje. Aj napriek tomu, že na stránke projektu sa uvádza, že je veľmi rýchle aj pri veľkých obrázkoch, v našich meraniach trvalo nahranie obrázka PANORAMA 26 sekúnd, a následné posúvanie nebolo možné, nakoľko sa obrázok prekresľoval niekoľko sekúnd.

http://foliotek.github.io/Croppie/ (B) (30. 4. 2016)
je ďalšie jQuery rozšírenie, ktoré slúži na úpravu obrázkov. Podporuje približovanie aj stredným tlačidlom myši. Samotná knižnica nepodporuje nahrávanie obrázkov. Umožňuje úpravu obrázkov už nahraných, a to takým spôsobom, že ich vykreslí do elementu <*img*>, ktorým umožňuje posúvať a pri uložení prekreslí výrez obrázka do canvasu, ktorý vyexportuje. Vzorové použitie však ukazuje aj prípad, kedy knižnica dokáže spracovať nahrané dáta. V tom prípade celý obrázok vykreslí do elementu <*canvas*>. Následne s ním pracuje rovnako ako v prvom prípade s elementom <*img*>. Nahrávanie obrázka PANORAMA trvalo priemerne 13 sekúnd (od odoslania po zobrazenie obrázka), ale na jeho plynulé posúvanie bolo potrebné čakať ešte zhruba minútu a pol. Myslíme si, že je to spôsobené tým, že vytvorený element <*canvas*> mal rovnaké rozlíšenie ako originálny obrázok, čo značne predlžovalo dobu vykreslenia v prehliadači.

http://andyvr.github.io/picEdit/ (C) (30. 4. 2016)
umožňuje nahrávanie obrázka cez modálne okno. Obrázok následne vykreslí ako pozadie elementu <*div*>. Približovanie mení parametre[1], akými sa toto pozadie zobrazuje. Následné ukladanie prebieha opäť cez prekreslenie do elementu <*canvas*>, z ktorého je výsek vyexportovaný. Nahranie obrázka PANORAMA bolo celkom rýchle, avšak posúvanie nebolo možné a približovanie trvalo dlhú dobu.

1. používajú sa CSS parametre *background-size* a *background-position*

http://codecanyon.stbeets.nl/ (D) (30. 4. 2016)
je platené (11$) rozšírenie jQuery umožňuje nahrávanie obrázkov aj cez modálne okno, aj cez *Drag&Drop*. Nahraný obrázok vykresľuje v elemente <*img*>, ktorým umožňuje posúvať, a následné uloženie obrázka spôsobí prekreslenie do elementu <*canvas*>. Uloženie je potrebné spraviť ručne. Pri nahratí obrázka ABSTRACT posúvanie mierne sekalo, pri nahratí obrázka PANORAMA nebolo možné.

Tieto vybrané knižnice sme porovnali v nasledujúcich vlastnostiach:

1. Závislosti na knižniciach tretích strán.
2. Podpora odosielania obrázka cez formulár (element <*form*>).
3. Nahrávanie viacerých obrázkov.
4. Podpora nahrávania cez *Drag&Drop*.
5. Možnosť stanoviť orezanie (napríklad posúvaním obrázka).
6. Podpora približovania.
7. Čas nahratia obrázka ABSTRACT[2].
8. Čas nahratia obrázka PANORAMA[2].

Nepodarilo sa nám nájsť riešenie, ktoré by fungovalo obdobne ako naše (používalo po celú dobu len element <*canvas*>). Každé z vybraných riešení k problematike nahrávania a zobrazovania obrázka pristupuje inak. Žiadne z vybraných riešení nebolo schopné vysporiadať sa so skutočne veľkým obrázkom, zatiaľčo naše riešenie ho nie len nahralo v rekordne krátkom čase, ale zároveň hneď umožňovalo aj jeho plynulú manipuláciu.

2. Test prebiehal na ultrabooku ASUS Zenbook UX31A (Intel i7-3517U, 4GB RAM, Microsoft Windows 10, Chrome 50. Časy boli odčítané z videa, ktoré bolo nahrávané počas nahrávania obrázkov.)

	Naša knižnica	A	B	C	D
1	žiadne	jQuery	jQuery	žiadne	jQuery
2	áno	nie	nie	nie	áno
3	áno	nie	nie	nie	nie
4	áno	nie	nie	nie	áno
5	áno	áno	áno	áno	áno
6	áno	áno	áno	áno	áno
7	0,9s	3,6s	2,5s	1,3s	2,9s
8	3,6s	23,6s	min. 17,6s	7,6s	15,6s

Tabuľka 5.1: Porovnanie našej knižnice s inými vybranými riešeniami.

6 Záver

V tejto práci sme najskôr predstavili najpoužívanejšie spôsoby na nahrávanie obrázkov v súčasnosti. Následne sme poukázali na ich nedostatky, ktoré boli základom pre naše riešenie. Stanovili sme si ambiciózne ciele a následne sme naše riešenie vytvárali tak, aby ich spĺňalo. Chceli sme vytvoriť knižnicu, ktorá bude nezávislá od iných knižníc a frameworkov. To sa nám použitím iba JavaScriptu podarilo. Vďaka zapuzdreniu, ktoré poskytujú webové komponenty nie je možné, aby tento JavaScript zasahoval do ostatných častí aplikácie. Takisto implementácia knižnice ako webového komponentu spôsobuje, že naše riešenie je veľmi jednoduché na použitie. Podporuje nahrávanie obrázkov cez *Drag&Drop*, ale aj modálne okno a rovnako podporuje aj nahrávanie viacerých obrázkov. Všetko toto je možné spraviť aj cez mobilné zariadenia. Obrázky je možné nahrávať bežnou cestou cez formulár (v elemente <*form*>), alebo cez *XMLHttpRequest*, a teda bez obnovenia stránky. Užívateľ ale v oboch prípadoch vidí nahrávaný obrázok ešte pred odoslaním a má možnosť upraviť orezanie. Keďže si uvedomujeme, že použitie elementu <*canvas*> na zobrazenie obrázka s možnosťou prístupu až k jednotlivým pixelom otvára dvere obrovskému množstvu možností (napríklad približovanie, detekcia tvárí, úprava farebnej škaly, inteligentné orezanie), ktorých implementácia nie je v našich silách, vymysleli a popísali sme jednoduchý, ale elegantný spôsob, ako tieto rozšírenia písať, použiť a kombinovať, a to či už globálne alebo pre konkrétne elementy. V neposlednom rade sme použitie zdokumentovali a s celou knižnicou zverejnili v službe *Github*.

Po predstavedstavení a rozbore nášho riešenia v kapitole 4, sme ho v ďalšej kapitole otestovali a porovnali s vybranými inými riešeniami, ktoré sú založené na podobných princípoch. Na konci tejto kapitoly ešte uvádzame odporúčané použitie a uvažujeme, akým spôsobom by sa mohlo naše riešenie ďalej posúvať.

6.1 Prínosy a odporúčané použitie

Knižnica umožňuje okamžite zobraziť obrázok bez potreby serveru. Všetky zmeny sa dejú v prehliadači, a teda skúsenosť používateľa nie

je zaťažená načítavaním. Odporúčame ju používať na projekty, kde je známe finálne rozlíšenie zobrazovaného obrázka (napr. blogy, internetové obchody, firemné stránky, profilové obrázky) a nie je potrebné si uchovávať pôvodný veľký obrázok. Hoc je možné použiť knižnicu aj v takýchto prípadoch, strácajú sa ďalšie prínosy knižnice. (A to) Napríklad skutočnosť, že sa odosiela zmenšená a orezaná verzia, ktorá tak menej zaťažuje sieť a v prípade spoplatnených mobilných dát šetrí peniaze používateľa. Tiež sa stráca výhoda, že nie je potrebné implementovať zmenšovanie a orezávanie obrázka na serveri. V prípade odosielania obrázka ako *Base64* reťazca (štandardný spôsob) je dokonca množstvo prenášaných dát asi o 30% väčšie.

6.2 Nápady na vylepšenie

Vzhľadom na obmedzenia, na ktoré sme počas vývoja narazili, dúfame, že našu prácu bude možné v budúcnosti vylepšiť v nasledujúcich oblastiach:

Zrkadlenie časti obsahu Custom Elementu do DOM. Ako sme v kapitole 4.4.4 uviedli, nezhoda medzi tvorcami prehliadačov spôsobuje, že v dobe písania sa už neodporúča na zrkadlenie využiť element <*content*>, ale ešte sa neodporúča používať element <*slot*>. Po štandardizovaní štandardu a následnom zapracovaní do knižnice by to umožnilo odosielanie viacerých obrázkov vo formulári (elemente <*form*>).

Odosielanie obrázkov ako *Blob*. Vzhľadom k tomu, že ukladanie obrázka ako *Base64* reťazec zväčšuje jeho veľkosť o 30%, ideálne by bolo ukladať a prenášať ho ako *Blob*. To je v dobe písania tejto práce podporované len v prehliadači Mozilla Firefox, IE 11 a Google Chrome ohlásil podporu v budúcej verzii (50) [24].

Literatúra

[1] Mozilla Developer Network, "<template>." https://developer.mozilla.org/en-US/docs/Web/HTML/Element/template. cit. 2016-04-13.

[2] http://caniuse.com, "Can I use... Support tables for HTML5, CSS3, etc." http://caniuse.com. cit. 2016-04-13.

[3] J. Ito, "Steve Jobs and Bill Gates."

[4] W3C, "HTML 3.2 Reference Specification." http://www.w3.org/TR/REC-html32, 1997. cit. 2014-11-15.

[5] Mozilla Developer Network, "FormData." https://developer.mozilla.org/en-US/docs/Web/API/FormData. cit. 2015-11-14.

[6] Mozilla Developer Network, "XMLHttpRequest." https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest. cit. 2015-11-14.

[7] Mozilla Developer Network, "progress." https://developer.mozilla.org/en-US/docs/Web/Events/progress. cit. 2015-11-14.

[8] C. David Goldman, "The beginning of the end for Adobe's Flash." http://money.cnn.com/2011/11/10/technology/adobe_flash/, 11 2011. cit. 2016-04-13.

[9] S. Jobs, "Thoughts on Flash." http://www.apple.com/hotnews/thoughts-on-flash/, 2010. cit. 2014-11-15.

[10] BBC, "Adobe Flash Player exits Android Google Play store." http://www.bbc.com/news/technology-19267140, 2015. cit. 2014-11-15.

[11] IDC, "Smartphone OS Market Share, 2015 Q2." http://www.idc.com/prodserv/smartphone-os-market-share.jsp, 2015. cit. 2014-11-15.

[12] Mozilla Developer Network, "Web components." https://developer.mozilla.org/en-US/docs/Web/Web_Components. cit. 2015-11-21.

[13] O. S. ASA, "Opera Software ASA - Opera version history." http://www.opera.com/docs/history/. cit. 2016-04-13.

[14] WebComponents.org, "webcomponents.js." https://github.com/WebComponents/webcomponentsjs. cit. 2016-04-12.

[15] Mozilla Developer Network, "Shadow dom." https://developer.mozilla.org/en-US/docs/Web/Web_Components/Shadow_DOM. cit. 2016-04-13.

[16] Mozilla Developer Network, "Canvas." https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API. cit. 2015-11-21.

[17] Mozilla Developer Network, "Window.requestanimationframe()." https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame. cit. 2015-11-21.

[18] Mozilla Developer Network, "Filereader." https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise. cit. 2015-11-21.

[19] Mozilla Developer Network, "Promise." https://developer.mozilla.org/en-US/docs/Web/API/FileReader. cit. 2015-11-21.

[20] K. Finley, "What Exactly Is GitHub Anyway?." http://techcrunch.com/2012/07/14/what-exactly-is-github-anyway/. cit. 2016-04-12.

[21] Mozilla Developer Network.

[22] W. Page, "The state of Web Components." https://hacks.mozilla.org/2015/06/the-state-of-web-components/, year = "2015", month = "6", note = cit. 2016-04-24.

[23] Mozilla Developer Network, "<content>." `https://developer.mozilla.org/en-US/docs/Web/HTML/Element/content`. cit. 2016-04-24.

[24] Mozilla Developer Network.