

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



## **Podakovanie**

This is the acknowledgement for my thesis, which can span multiple paragraphs.

## **Zhrnutie**

This is the abstract of my thesis, which can span multiple paragraphs.



## **Klíčové slová**

keyword1, keyword2, ...



# Obsah

1	<b>Súčasn� metody nahr�vania obr�zkov</b>	1
1.1	HTML t�g <input>	1
1.2	FormData a XMLHttpRequest	2
1.3	Rie�enia zalo�en� na Adobe® Flash® a Microsoft® Silverlight®	3
2	<b>Nev�hody s�časn�ch metod</b>	5
2.1	Podpora mobiln�ch zariaden�	5
2.2	Zbyto�n� s�bory na serveri	5
2.3	Orezanie obr�zka	5
2.4	Pren��anie zbyto�n�ch d�t	6
3	<b>Vytvorenie kni�nice na nahr�vanie obr�zkov</b>	7
3.1	Ciele	7
3.2	Rie�enie	7
3.2.1	Predstavenie	7
3.2.2	In�tal�cia	8
3.3	Pou�it� technol�gie, slu�by a postupy	11
3.3.1	Webov� komponenty	11
3.3.2	Canvas	12
3.3.3	DragDrop	13
3.3.4	File Reader	13
3.3.5	Promise	13
3.3.6	TypeScript	13
3.3.7	Mocha	14
3.3.8	GitHub	14
3.4	Postup nahr�vania a nastavenia	14
3.4.1	Inicializ�cia kni�nice	14
3.4.2	Zvolenie obr�zka	14
3.4.3	Zmen�enie a orezanie	15
3.4.4	Pos�vanie obr�zka	17
3.4.5	Odoslanie obr�zka	17
3.5	Nahr�vanie viacer�ch obr�zkov	19
3.6	Roz��iritel�nos�	19
3.6.1	Syst�m roz��iren�	19
3.6.2	Cibu�ov� efekt	20
3.6.3	Aplikovanie roz��irenia	21
3.6.4	Vzorov� roz��irenie	21

3.7	<i>Spracovanie obrázka na PHP serveri</i>	23
3.8	<i>Dokumentácia</i>	23
4	<b>Validácia riešenia</b>	25
4.1	<i>Podpora mobilných zariadení</i>	25
4.2	<i>Jednotkové testy</i>	25
4.3	<i>Porovnanie s ďalšími riešeniami</i>	25
5	<b>Záver</b>	27
5.1	<i>Odporúčané použitie</i>	28
5.2	<i>Nápady na vylepšenie</i>	28

## **Zoznam tabuliek**

- 3.1 Podpora webových komponent v prehliadačoch  
pre počítače [1] [2] 11
- 3.2 Podpora webových komponent v mobilných  
prehliadačoch [1] [2] 12



## Zoznam obrázkov

- 1.1 Schéma nahrávania obrázka pomocou *<input>* elementu 1
- 1.2 Schéma nahrávania obrázka pomocou *<input>* elementu v kombinácii s *FormData* a *XMLHttpRequest* 3
- 3.1 Schéma nahrávania obrázka pomocou *<x-cupe>* elementu 10
- 3.2 Nákres možného scenára nahrávaného obrázka 22



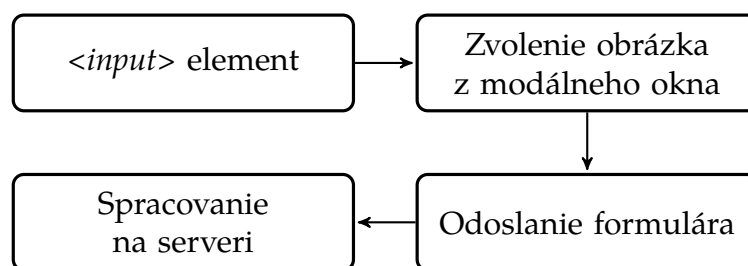


# 1 Súčasné metódy nahrávania obrázkov

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.

## 1.1 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* [3]. Umožňuje nahrávanie súborov a teda aj obrázkov ako súčasť zaslaného formulára, a teda sa vždy musí vyskytovať uprostred tágú `<form>`. Scénar nahrávania prebieha nasledovne:



Obr. 1.1: 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 *dragdrop*

### 1.2 FormData a XMLHttpRequest

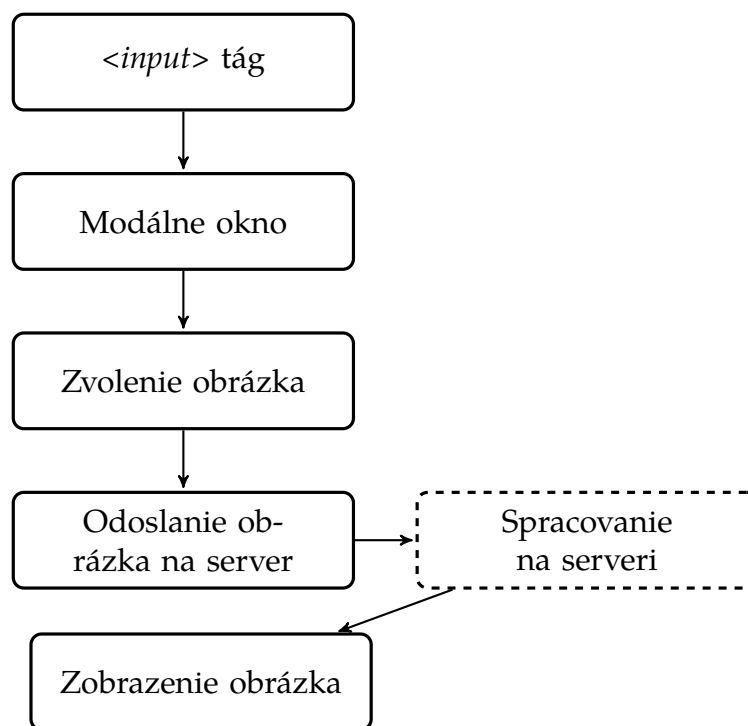
*XMLHttpRequest Level 2* pridáva nové rozhranie *FormData*, ktoré umožňuje vytvoriť reprezentáciu formulára v tvare kľúč-hodnota, čím umožňuje zaslať formulár pomocou *XMLHttpRequest* [4]. *XMLHttpRequest* je API, ktoré umožňuje prenášať dáta medzi prehliadačom a serverom bez toho aby nastal refresh [5].

Vďaka *FormData* je možné odstrániť limitáciu, kde `<input type="file">` musí byť vložený do formulára (`<form>`). Použitím *XMLHttpRequest* eventu *onprogress* je možné sledovať proces nahrávania [6]. 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 sleduje proces nahrávania. Keď je obrázok plne nahratý 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 ku 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 aplikáciach, kde je refresh nežiadúci
- Počas nahrávania je možné zobrazíť proces v percentách
- Po nahratí obrázka na server, je možné obrázok užívateľovi zobrazíť. A teda ho vidí pred potvrdením formulára
- Podpora *dragdrop*



Obr. 1.2: Schéma nahrávania obrázka pomocou `<input>` elementu v kombinácii s `FormData` a `XMLHttpRequest`

### 1.3 Riešenia založené na Adobe® Flash® a Microsoft® Silverlight®

V dobe písania tejto práce boli stále veľmi rozšírené riešenia založené na technológií Adobe® Flash® (ďalej spomínaný už len ako Flash). Flash pôvodne navrhnutý pre tvorbu multimedialného obsahu - vektorovej grafiky, animácií, hier pre prehliadač - sa stal populárnym aj na tvorbu väčších aplikácií v prehliadači, pracovanie so vstupom 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 nepoužívali 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 technológiach 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 2.1). 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[7].

## 2 Nevýhody súčasných metód

### 2.1 Podpora mobilných zariadení

V roku 2010, Steve Jobs, spoluzakladateľ a v tom čase aj výkonný riaditeľ Applu vydal vyhlásenie *Thoughts on Flash*[8], v ktorom vysvetľuje prečo zariadenia iPhone, iPod a iPad nepodporujú *Adobe Flash*. Kritizuje *Adobe Flash* z uzavrenosti, vysokej systémovej záťaže, slabej bezpečnosti a chýbajúcej podpore dotykových zariadení. BBC v roku 2012 informovala[9], 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[10] nepodporujú *Adobe Flash*.

### 2.2 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 obrázku na serveri (napríklad nesprávny orez, pozri 2.3), môže zmeniť svoje rozhodnutie. Takto vznikajú na serveri súbory - obrázky, ktoré sa nikdy nepoužijú. Jedno z riešení tohto problému je pravidelné mazanie nevyužívaných obrázkov.

### 2.3 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 XX (TODO), 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.

## 2. NEVÝHODY SÚČASNÝCH METÓD

---



### 2.4 Prenášanie zbytočných dát

// TODO

## 3 Vytvorenie knižnice na nahrávanie obrázkov

### 3.1 Ciele

Ciele, ktoré sme stanovili pri vytváraní knižnice majú za cieľ maximalizovať pohodlie užívateľa, minimalizovať zaťaženie pre tvorcov stránky a mala 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 *DragDrop*,
- byť jednoduchá na implementovanie
- byť dobre rozšíriteľná o nové funkcie
- podporovať nahrávanie bez refreshu
- podporovať mobilné zariadenia
- podporovať nahrávanie viacerých obrázkov
- byť dobre zdokumentovaná

### 3.2 Riešenie

#### 3.2.1 Predstavenie

Najskôr sme plánovali využiť *jQuery*, ale kvôli splneniu prvého cieľa (býť nezávislý 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 multiuploadu - `<x-cupe-gallery>`. Oba vychádzajú z rovnakého HTML, pričom `<x-cupe-gallery>` interne využíva `<x-cupe>`, preto sa ďalej v tejto kapitole zameráme na element

### 3. VYTVORENIE KNIŽNICE NA NAHRÁVANIE OBRÁZKOV

---

`<x-cupe>`. Nahrávanie viacerých obrázkov je bližšie popísané v kapitole 3.5.

Scénar nahrávania môže vyzeráť takto:

1. Užívateľ buď pomocou DragDrop 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 obrázkom, pokračuje prvým bodom
4. Pokiaľ užívateľ nie je spokojný s orezaním, držaním myši posúva obrázok, dokým nie je spokojný
5. Keď je užívateľ spokojný s obrázkom aj jeho orezaním odošle formulár, a až vtedy sa zmenšaná 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">
```

---

Element `<canvas>` sa stará o zobrazenie obrázka, a 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 popísanie, ako knižnica funguje vrátane využitia jednotlivých elementov je ďalej popísané v sekcii 3.4.

#### 3.2.2 Inštalácia

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



### 3. VYTVORENIE KNIŽNICE NA NAHRÁVANIE OBRÁZKOV

---

---

```
<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:

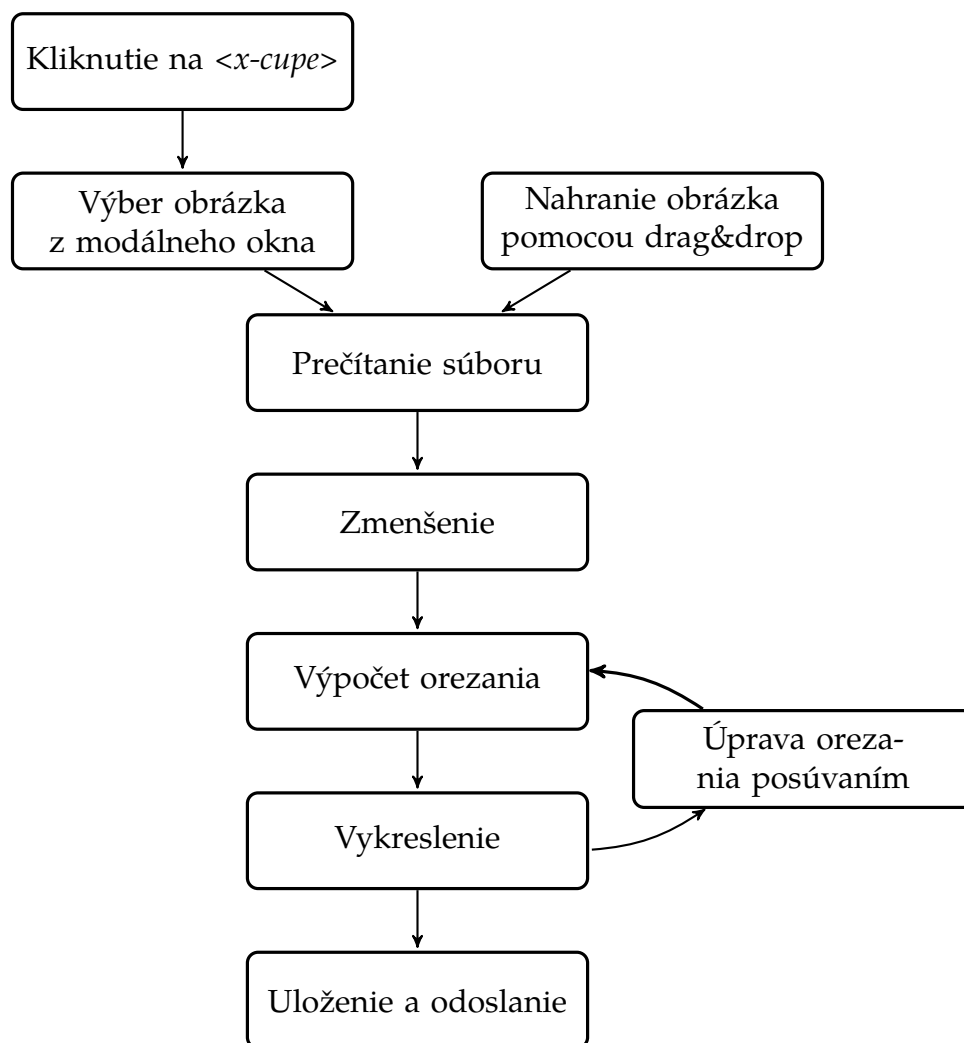
---

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

---

### 3. VYTVORENIE KNIŽNICE NA NAHRÁVANIE OBRÁZKOV

---



Obr. 3.1: Schéma nahrávania obrázka pomocou `<x-cupe>` elementu

### 3.3 Použité technológie, služby a postupy

#### 3.3.1 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 komponent užívateľského rozhrania. Keďže sú <sup>1</sup> 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. [?]

V dobe písania tejto práce boli webové komponenty podporované plne len v *Chrome* a v pokročilom stave vývoja v prehliadači *Firefox*<sup>2</sup>. Podrobnejší popis podpory uvádzame v tabuľkách 3.1 a 3.2<sup>3</sup>. Vzhľadom k tomuto stavu v práci používame náhradu (tzv. "polyfill") *web-components.js* [12]. Tá poskytuje podporu pre webové komponenty aj do všetkých aktuálnych dnes používaných prehliadačov, ktoré ju ešte nemajú zabudované. Výnimku tvorí *Internet Explorer 10*, ktorý nepodporuje *Custom Elements* a *HTML Imports*. [12]

*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 odstavce, 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* [11]

	IE 11	Edge	Chrome	Safari	Firefox
Templates	Nie	13+	26+	7.1+	22+
HTML Imports	Nie	Nie	47+	Nie	Nie <sup>2</sup>
Custom Elements	Nie	Nie	47+	Nie	Nie <sup>2</sup>
Shadow DOM	Nie	Nie	47+	Nie	Nie <sup>2</sup>

Tabuľka 3.1: Podpora webových komponent v prehliadačoch pre počítače [1] [2]

### 3. VYTVORENIE KNIŽNICE NA NAHRÁVANIE OBRÁZKOV

	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 3.2: Podpora webových komponent v mobilných prehliadačoch [1] [2]

vanie v prípade, že bol element vytvorený, vložený do *DOM*<sup>4</sup>, odstránený z *DOM* alebo sa zmenili atribúty. [?]

*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 *JavaScriptu* do webovej komponenty. A robí to tak, že tieto nebudú súčasťou *DOM* alebo hlavného dokumentu. Hlavná výhoda spočíva predovšetkým v štylovaní, kde použitie *Shadow DOM* zaručuje, že *CSS* iných častí stránky neovplyvňujú vzhľad komponentny, a komponenta nemôže ovplyvniť vzhľad iných častí stránky. [13]

#### 3.3.2 Canvas

Canvas je nový tág *HTML5*, ktorý umožňuje pomocou JavaScriptu vykresľovať grafy, upravovať fotografie, animovať a dokonca aj spracovať videa. Využíva ho aj *WebGL* na hardvérovo akcelerovanú 3D grafiku na webe[14].

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 čierneho-bielej verzie). Keď užívateľ presúva obrázok, kvôli

---

4. Document Object Model

orezaniu, prekresľujeme daný výrez do `<canvas>` pomocou `requestAnimationFrame`[15], pre maximálnu plynulosť.

#### 3.3.3 DragDrop

#### 3.3.4 File Reader

File Reader umožňuje čítať obsah súborov vo webovom prostredí. Tie môžu byť načítané buď pomocou tagu `<input>` alebo pomocou `DragDrop` (pozri

Použili sme ho práve pre prečítanie zvolených súborov, ktoré sme následne zobrazili v `<canvas>` (pozri 3.3.2.

#### 3.3.5 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[16]. V silno asynchrónnej aplikácii zabraňuje tzv. "callback hell", kde množstvo callbackov bráni zrozumiteľnosti a ladeniu kódu.

Promise sme používame miesto predávania callbackov, predovšetkým pri čítaní užívateľovho obrázku.

#### 3.3.6 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 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.

#### 3.3.7 Mocha

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

#### 3.3.8 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 [17].

### 3.4 Postup nahrávania a nastavenia

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

#### 3.4.1 Inicializácia knižnice

#### 3.4.2 Zvolenie obrázka

Užívateľ môže obrázok zvoliť dvoma spôsobmi - výberom z modálneho okna alebo pomocou *DragDrop*. V prípade nahrávania obrázka cez modálne okno, užívateľ musí kliknúť na `<x-cupe>` element, do ktorého chce obrázok nahráť. 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 3.3.4) 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 *DragDrop* a teda označenie obrázka, jeho presunutie ponad cieľový element a následne 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*.

#### 3.4.3 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). Pre lepšie porozumenie uvádzame aj obrázok. // TODO obrázok

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 spravca 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.

### 3. VYTVORENIE KNIŽNICE NA NAHRÁVANIE OBRÁZKOV

---

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. obrázkove galérie, pričom chceme, aby obrázky mali pevnú šírku a ľubovoľne dlhú dĺžku. Ak chceme variabilnú šírku, je potrebné nastaviť atribút *width* elementu `<x-cupe>` na hodnotu `-1`. V prípade, že chceme variabilnú dĺžku, 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 zmenšenej dĺžke 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 3.4.4) 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 obrázku 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 ostrejši 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 najrychlejšie, 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 manipulácii ho nebolo potrebné opäť zmenšovať.

#### 3.4.4 Posúvanie obrázka

Pokiaľ je povolené orezanie obrázka, je možné toto orezanie meniť posúvaním obrázka. Aby sme odlíšili kliknutie, kedy má vyskočiť modálne okno s možnosťou 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 prirába k orezaniu obrázka. Nakoniec už zmenšený obrázok s novým nastavením orezania vykreslíme s využitím *requestAnimationFrame*. Posúvanie obrázka sa dá vypnúť nastavením atribútu *allow\_move* na hodnotu *false*.

Vzhľadom k tomu, že ukladanie (pozri 3.4.5) je časovo náročná operácia, počas posúvania k nemu nedochádza. Keď ale užívateľ posúvanie ukončí, obrázok je uložený.

#### 3.4.5 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">* (ďalej označovaný input), 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 príbežne ukladá - po načítaní obrázka a potom, ako skončí jeho posúvanie (pozri 3.4.4).

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 me-

### 3. VYTVORENIE KNIŽNICE NA NAHRÁVANIE OBRÁZKOV

---

tódu `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*<sup>5</sup>. V prípade formátov *JPG* a *WEBP* je tiež možné nastaviť kvalitu v rozsahu od 0 (0

---

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

---

Listing 3.1: 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 spôsobami:

- *Custom Element* využíva *Shadow DOM* na internú reprezentáciu, ale tiež môže reflektovať aj obsah, ktorý je vložený dovnútra elementu v DOM na konkrétne miesto v *Shadow DOM*. Problémom však je, že sa tvorcovia prehliadačov nevedia dohodnúť na tom, ako by to malo fungovať [18]. Kým v pôvodnej špecifikácii sa objavuje nový element `<content>`, MDN<sup>6</sup> ho už neodporúča používať [19]. 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 *CustomElementu*, ale nie do *Shadow DOM*. Je to jednoduchý trik. Keďže takýto element je súčasťou DOM, ale nie je súčasťou *ShadowDOM* prehliadač nevie ako ho

---

5. Nemusí byť podporovaný v každom prehliadači

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

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 nechcený jav.

## 3.5 Nahrávanie viacerých obrázkov

Nahrávanie viacerých obrázkov je možné pomocou párového tagu `<x-cupe-gallery>`. Ten interne využíva elementy `<x-cupe>`, ktoré vytvára vždy pri zvolení nového obrázka, či už pomocou modálneho okna alebo pomocou *DragDrop*. 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.

## 3.6 Rozšíriteľnosť

Nakoľko naša knižnica používa na spracovanie a zobrazovanie obrazu tag `<canvas>`, využíva a aj poskytuje možnosť pracovať s obrazom na úrovni jednotlivých pixelov. To umožňuje vytváranie pokročilých rozšírení, ktoré napríklad dokážu zmeniť farebné odtiene, využívať detekciu hrán, rozpoznávanie tvárí a podobne.

### 3.6.1 Systém rozšírení

Predstavme si dve rozšírenia - "zoom"(rozšírenie umožňujúce zväčšovať a zmenšovať obraz) a "black&white"(rozšírenie, ktoré prevedie obraz do čierneho-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" a "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".

#### 3.6.2 Cibuľový efekt

Pre lepšie vysvetlenie uvádzame vzorku kódu so vzorového rozšírenia "zoom"(pozri 3.6.4, na ktorom následne vysvetlíme, "cibuľový efekt" jeho výhody.

---

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

---

Listing 3.2: 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 doplnujúci kód.

Je potrebné si uvedomiť, že metóda *readAndDrawImage* ostáva zmenená a keď k nej bude pristupovať ďalšie rozšírenie (napr. "black&white" popísaný v 3.6.1), 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`.

#### 3.6.3 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 3.6.4) 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
makeCupeZoomable();

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

---

#### 3.6.4 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 zobrazovacou 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 tlačidlom myši mení pomer, akým sa pôvodný obrázok zmenší predtým, než sa vykreslí.

Pre lepšie porozumenie opíšeme postup pre vzorový scénar popísaný obrázkom 3.2.

Inicializácia. Na začiatku sa určí minimálne priblíženie (vtedy vidíme najväčšiu časť obrázka) na hodnotu 1, maximálne priblí-

### 3. VYTVORENIE KNIŽNICE NA NAHRÁVANIE OBRÁZKOV

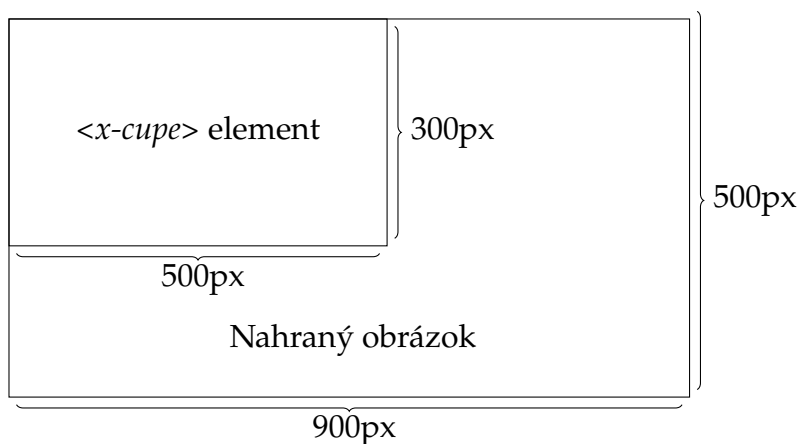
---

ženie 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 prenášobia 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 zoomu.



Obr. 3.2: Nákres možného scenára nahrávaného obrázka

### 3.7 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, vzorovbé 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 predpokladá s odoslaním formulára (elementu *<form>*), kde sa obrázky odosielaajú 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.

### 3.8 Dokumentácia





## **4 Validácia riešenia**

### **4.1 Podpora mobilných zariadení**

### **4.2 Jednotkové testy**

### **4.3 Porovnanie s ďalšími riešeniami**



## 5 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 poradilo. Vďaka zapuzdreniu, ktoré poskytujú webové komponenty nie je možné, aby tento JavaScript zasahoval do ostatných častí aplikácie. Takisto implementácia knižnice ako webovej komponenty spôsobuje, že naše riešenie je veľmi jednoduché na použitie. Podporuje nahrávanie obrázkov cez modálne okno, ale aj modálne okno a rovnako podporuje aj nahrávanie viacerých obrázkov. Všetko toto je možné spraviť aj cez mobilné zariadenia. Obrázky je možné nahrávať či bežnou cestou cez formulár (v elemente `<form>`) alebo cez `XMLHttpRequest` a teda bez refreshu. 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 zoomovanie, detekcia tvári, úprava farebnej škaly, inteligentné orezanie), ktorých implementácia nie je v našich silách, vymysleli a popísali sme jednoduchý, zato 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 predstavení a rozbere nášho riešenia v kapitole 3, 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.

## 5. ZÁVER

---

### 5.1 Odporúčané použitie

### 5.2 Nápady na vylepšenie

## 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] W3C, "HTML 3.2 Reference Specification." <http://www.w3.org/TR/REC-html32>, 1997. cit. 2014-11-15.
- [4] Mozilla Developer Network, "FormData." <https://developer.mozilla.org/en-US/docs/Web/API/FormData>. cit. 2015-11-14.
- [5] Mozilla Developer Network, "XMLHttpRequest." <https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest>. cit. 2015-11-14.
- [6] Mozilla Developer Network, "progress." <https://developer.mozilla.org/en-US/docs/Web/Events/progress>. cit. 2015-11-14.
- [7] C. David Goldman, "The beginning of the end for Adobe's Flash." [http://money.cnn.com/2011/11/10/technology/adobe\\_flash/](http://money.cnn.com/2011/11/10/technology/adobe_flash/), 11 2011. cit. 2016-04-13.
- [8] S. Jobs, "Thoughts on Flash." <http://www.apple.com/hotnews/thoughts-on-flash/>, 2010. cit. 2014-11-15.
- [9] BBC, "Adobe Flash Player exits Android Google Play store." <http://www.bbc.com/news/technology-19267140>, 2015. cit. 2014-11-15.
- [10] IDC, "Smartphone OS Market Share, 2015 Q2." <http://www.idc.com/prodserv/smartphone-os-market-share.jsp>, 2015. cit. 2014-11-15.
- [11] O. S. ASA, "Opera Software ASA - Opera version history." <http://www.opera.com/docs/history/>. cit. 2016-04-13.

## LITERATURA

---

- [12] WebComponents.org, “webcomponents.js.” <https://github.com/WebComponents/webcomponentsjs>. cit. 2016-04-12.
- [13] Mozilla Developer Network, “Shadow dom.” [https://developer.mozilla.org/en-US/docs/Web/Web\\_Components/Shadow\\_DOM](https://developer.mozilla.org/en-US/docs/Web/Web_Components/Shadow_DOM). cit. 2016-04-13.
- [14] Mozilla Developer Network, “Canvas.” [https://developer.mozilla.org/en-US/docs/Web/API/Canvas\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API). cit. 2015-11-21.
- [15] Mozilla Developer Network, “Window.requestAnimationFrame().” <https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame>. cit. 2015-11-21.
- [16] Mozilla Developer Network, “Promise.” <https://developer.mozilla.org/en-US/docs/Web/API/FileReader>. cit. 2015-11-21.
- [17] K. Finley, “What Exactly Is GitHub Anyway?” <http://techcrunch.com/2012/07/14/what-exactly-is-github-anyway/>. cit. 2016-04-12.
- [18] 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.
- [19] Mozilla Developer Network, “<content>.” <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/content>. cit. 2016-04-24.