Knihovna pro validaci vstupních textů webového formuláře

Bakalářská práce

Studijní program: B2646 – Informační technologie
Studijní obor: 1802R007 – Informační technologie

Autor práce: Jakub Vejr
Vedoucí práce: Ing. Jiří Hnídek, Ph.D.

Liberec 2015/2016

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2015/2016**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jakub Vejr**
Osobní číslo:	**M13000144**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Knihovna pro validaci vstupních textů webového formuláře**
Zadávající katedra:	**Ústav nových technologií a aplikované informatiky**

Zásady pro vypracování:

1. Seznamte se současnými technologiemi pro validaci vstupních polí webových formulářů, které se provádí jak na straně webového serveru, tak na straně webového klienta.
2. Implementujte kontrolu vstupních textů webového formuláře, tak aby byl modulární a vhodně kombinoval možnost kontroly jak na straně klienta, tak na straně serveru s tím, že komunikace klienta s webovým musí být plně asynchronní.
3. Implementujte vhodnou sadu modulů pro kontrolu vstupních textů.
4. Navrhněte a implementujte pro každý modul vhodnou sadu testů (unit testy, integrační testy, apod.) a vytvořte k celému systému odpovídající dokumentaci.

Rozsah grafických prací: **dle potřeby**

Rozsah pracovní zprávy: **40 - 60 stran**

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] LOCKHART, J. Modern PHP. O'Reilly Media, 2015. ISBN:978-1-4919-0501-2

[2] FOGUS, M. Functional Javascript. O'Reilly Media, 2013. ISBN: 978-1-4493-6072-6

Vedoucí bakalářské práce: **Ing. Jiří Hnídek, Ph.D.**
Ústav nových technologií a aplikované informatiky

Datum zadání bakalářské práce: **20. října 2015**

Termín odevzdání bakalářské práce: **16. května 2016**

prof. Ing. Václav Kopecký, CSc.
děkan

prof. Dr. Ing. Jiří Maryška, CSc.
vedoucí ústavu

V Liberci dne 20. října 2015

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum

16.5.2016

Podpis

Poděkování

Tímto bych velice rád poděkoval vedoucímu Ing. Jiřímu Hnídkovi, Ph.D. za pomoc při zpracování této práce.

Abstract:

Cílem bakalářské práce je navrhnout aplikaci pro asynchronní kontrolu textů a vstupů webového formuláře s použitím nových technologií. Aplikace by měla mít možnost modulace a snadného použití. Největší problém je řešení validace na obou stranách zároveň, tzn. jak na straně klienta, tak na straně serveru. Tento problém často vede k nutnosti realizace aplikace pomocí dvou různých programovacích jazyků. Řešení tedy zkoumá nové možnosti v podobě isomorfních aplikací, což usnadní jak realizaci, tak údržbu.

Klíčová slova:

JavaScript, jQuery, Node.js, Socket.io, Browserify, Babel, ECMAScript 6, validace vstupů, validace textu.

Abstract:

The goal of this work is to design an application for asynchronous control input texts using novel technologies. Application should have an ability of modulation and be easy to use. The biggest problem is the solution of validation on both sides ie. the client and server side. This problem causes the necessity to use two the different programming languages in the same application. In the solution of this work will be explored new opportunities in the form of isomorphic applications, which will facilitate the implementation and maintenance.

Key words:

JavaScript, jQuery, Node.js, Socket.io, Browserify, Babel, ECMAScript 6, input validation, text validation.

Obsah

Seznam obrázku ... 8

Seznam zkratek ... 9

Úvod ... 10

1 Rešerše ... 11

2 Výběr technologie ... 12

2.1 Řešení PHP + Ajax + JavaScript ... 12

2.2 Řešení pomocí isomorfní aplikace ... 13

2.2.1 Bezpečnost isomorfních aplikací ... 13

2.2.2 Zvolení knihovny ... 14

2.3 Shrnutí ... 14

3 Návrh aplikace ... 15

3.1 Struktura Aplikace ... 15

3.2 Zvolení a návrh databáze ... 17

4 Validace ... 18

4.1 Důvody validace ... 18

4.2 Řešené validační problémy ... 19

4.3 Vulgarismy ... 20

4.3.1 Časová náročnost řešení ... 22

4.3.2 Ověření funkčnosti ... 23

4.3.3 Neuronové sítě ... 24

4.3.4 Zhodnocení ... 24

4.4 Univerzální třída pro validaci textového formuláře ... 25

4.4.1 Kontrola délky textu ... 25

4.4.2 Kontrola délky slova ... 25

4.4.4 Zabezpečení před XSS ... 27

4.4.5 Kontrola Emailu ... 27

4.4.6 Kontrola telefonního čísla ... 27

5 Využívané balíčky 28
5.1 NPM 28
5.2 Babel 28
5.3 Browserify 29
5.4 Node-Inspector 29
5.5 XSS 29
5.6 SQLite 3 30
5.7 Socket.io 31
6 Vytvoření serveru 32
6.1 Stáhnutí dat 32
6.2 Připravení knihoven a modulů pro vytvoření serveru 33
6.3 Vytvoření serveru pro naslouchání požadavků 34
6.4 Nastavení Socket.Io 37
7 Realizace klientské části 38
7.1 Vytvoření Toolbaru 38
7.2 Načtení dat přes Socket.IO 40
8 Používání aplikace 41
8.1 Klientská část 41
8.2 Server 43
9 Testování 44
Závěr 46
Seznam použité literatury 47

Seznam obrázku

Obrázek 1 Adresářová struktura ... 15
Obrázek 2 Ukázka databáze ... 17
Obrázek 3 Časová náročnost při detekování vulgarismu ... 22
Obrázek 4 Časová náročnost detekce vulgarismu s použitím přídavné funkce ... 22
Obrázek 5 Schéma popisující funkčnost Socket.Io ... 31
Obrázek 6 Ukázka Toolbaru ... 38
Obrázek 7 Ukázka úspěšné detekce webové stránky ... 41
Obrázek 8 Ukázka možné alternativní detekce ... 42
Obrázek 9 Ukázka konzolového výstupu ... 43
Obrázek 10 Ukázka výstupu qUnits při testování ... 44

Seznam zkratek

ES – ECMAScript

XSS – Cross-site scripting

Ajax – Asynchronous JavaScript and XML

IP – Internet Protocol

DoS – Denial of service

DDoS – Distributed denial of service

HTTP – Hypertext Transfer Protocol

SEO – Search Engine Optimization

Úvod

Bakalářská práce se zabývá implementací knihovny, která by měla umožňovat validaci textů a vstupů jak na straně klienta, tak na straně serveru. Z toho důvodu bude nutné zkoumat nejúčinnější možné řešení, které zajistí modularitu, ale i co nejsnazší údržbu kódu a pokud možno bez zbytečné duplicity kódu na obou stranách. Knihovna by měla umožňovat indikaci o porušení pravidla s možností vyrozumění uživatele. Dále by měla umožňovat opravu nalezeného problému. Indikace by se měla vyskytovat hlavně na straně klienta, zatímco ošetření problému na straně serveru. Řešení problémů by mělo probíhat zcela asynchronně. Aplikace by měla dále umožňovat vést si databázi s vybranými slovy, která budou potřeba pro řešení různých validačních problémů. Zároveň by se mělo zabránit případnému obcházení validačních pravidel. Aplikace by měla být plně modulární pro případné budoucí úpravy. Případné porušení pravidel by se uživateli mělo zobrazit pomocí toolbaru, který se zobrazí pod vstupem. Programátor využívající knihovnu se bude moci rozhodnout, zda bude chtít toolbar pod vstupem zobrazovat. Aplikace bude zaměřena na řešení validací v českém jazyce, jelikož v něm neexistuje žádné komplexní řešení. Na závěr je pro aplikaci třeba navrhnout vhodnou sadu testů.

1 Rešerše

Cílem rešerše je zjistit existenci podobných projektů a porovnat jejich funkcionalitu s nastaveným cílem a požadavky mého řešení.

Na začátku práce jsem zkoumal, zda neexistuje podobné řešení, ať už se zaměřením na český, nebo na cizí jazyk. Bylo zjištěno, že pro český jazyk neexistuje žádné komplexní řešení a ani žádné, které by se tomu blížilo. Knihovny se zaměřením na cizí jazyky existují, ale mají často velké nedostatky.

Jedním z příkladů jsou validace, které umožňuje samotná knihovna jQuery. Ta umožňuje validaci emailu nebo validaci telefonního čísla. Validace telefonního čísla je však zaměřena hlavně na americká čísla (americké předvolby atd.).

Hledání knihovny pro detekci vulgarismů se také nedočkalo uspokojivého výsledku. Žádné nalezené řešení totiž neumožnilo validaci na obou stranách, tzn. na straně klienta i serveru. Byla nalezená implementace v PHP s názvem BanBuilder, provádějící detekci slov pouze na straně serveru. Dále bylo zjištěno, že nezajišťuje 100% spolehlivost, což lze vzhledem k obtížnosti a možné úplné neřešitelnosti tohoto problému očekávat. Byla také nalezena podobná knihovna s názvem jQuery.ProfanityFilter, ale ta naopak umožňovala validaci pouze na straně klienta.

Obě z výše zmíněných knihoven nabízí pouze detekci v anglickém jazyce, kde jsou slova uchovávána pomocí slovníku. Pro český jazyk jsou tato řešení nepoužitelná. Výsledek zkoumání vede k závěru, že pro český jazyk neexistuje žádné komplexní řešení na straně klienta ani serveru. Ani v cizích jazycích neexistuje žádné řešení umožňující oboustrannou validaci s požadovanou lokalizací.

2 Výběr technologie

Při návrhu postupu vypracování aplikace se uvažovalo nad novými způsoby, ale zkoumalo se i řešení, které se dnes běžně používá. Jedno z dnes nejpoužívanějších řešení je kombinace PHP + JavaScript + Ajax. Další alternativa, která se začala využívat v posledních letech, jsou isomorfní aplikace, které umožňují využívat jeden programovací jazyk na obou stranách. I toto řešení má své nevýhody.

2.1 Řešení PHP + Ajax + JavaScript

Dnes je to jedno z nejpoužívanějších řešení. Jeho výhodou je, že je snadné pro vývoj složitějších aplikací a je široce používané. Řešení s využitím programovacího jazyka PHP je podporováno většinou hostingů a je známé většině programátorů. Při vypracovávání této práce se toto řešení v počátcích testovalo. Nevýhoda plynula z toho, že veškeré validace se musely posílat jako Ajax požadavek na server. Při větším počtu uživatelů vzniká zátěž na server. Kvůli rychlosti by se některé funkce, jako např. validace telefonního čísla, musely programovat dvakrát, tzn. na serveru i u klienta, aby nedošlo k zatěžování serveru. To může znamenat nevýhody i při testování a údržbě kódu. Při změně kódu algoritmu na jedné straně by se musela provést stejná změna i na straně druhé. To je tedy pro programátora značná časová zátěž. Všechny nevýhody tohoto řešení by se tedy daly shrnout do následujících bodů.

1. Obtížná údržba
2. Výkonová zátěž pro server při větším počtu uživatelů
3. Obtížné testování
4. Větší časová náročnost na implementaci
5. Nutná znalost minimálně dvou programovacích jazyků
6. Duplicita kódu

2.2 Řešení pomocí isomorfní aplikace

Tato technologie se dočkala rozmachu v posledních letech. Isomorfní aplikace je taková aplikace, která umožňuje programovat pomocí stejného jazyka na straně klienta i serveru. To umožňuje se vyhnout duplicitě kódu. Další výhodou tohoto řešení také je, že není potřeba zatěžovat neustále server požadavky od klientské části, na rozdíl od řešení pomocí Ajaxu. Prohlížeč jednoduše na začátku stáhne vybrané skripty. Z toho důvodu dojde k úspoře výkonu, protože se nemusíme neustále dotazovat serveru. Stejné funkce můžeme využívat na obou stranách. Snazší bude i údržba, jelikož stačí změnit pouze danou funkci a ke změně dojde na obou stranách. Výhodou také je, že k tomuto řešení je nutná znalost pouze jednoho programovacího jazyku.

I když se toto řešení zdá být výrazně lepší, i to ale nese značné nevýhody. Při složitějších aplikacích se toto řešení nemusí vyplatit z důvodů obtížné implementace na straně serveru a nízké úrovně objektového programování na straně JavaScriptu. Tento nedostatek lze z části vyřešit používáním nového standartu ECMAScript 6 [2], ale i zde jsou stále nedostatky, i když už obsahuje základní syntaxi pro objektové programování.

2.2.1 Bezpečnost isomorfních aplikací

Ze začátku vývoje byly isomorfní aplikace náchylné na DoS útoky [16], ale s dobrou podporou ze strany vývojářů se neustále vydávají nové bezpečnostní záplaty. Např. Node.js vydává minimálně každý rok bezpečnostní aktualizace chránící server před DoS útoky. Mají však stále problémy s DDos útoky, ale těm nedokáží účinně zabránit téměř žádné dnes používané servery.

2.2.2 Zvolení knihovny

Pro implementaci serveru se nejčastěji využívá JavaScriptová knihovna Node.js, která je také použita v tomto řešení. V praxi to vypadá tak, že vytvoříme server, který bude naslouchat na určitém portu. Ten bude přijímat požadavky a přizpůsobovat jim odpovědi.

2.3 Shrnutí

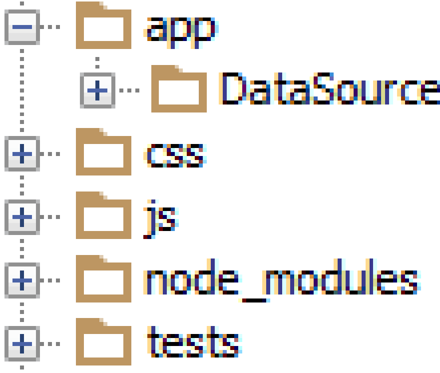
Po vyhodnocení obou hlavních technologií jsem se rozhodl zvolit Node.js. Nejedná se sice o často používané řešení, ale pro případ oboustranné validace je jednoznačně výhodnější. Umožní používat stejné funkce u klienta i serveru a usnadní tak testování a případnou modulaci. Je ale jasné, že bude potřeba spousta přídavných knihoven, jako např. knihovny pro asynchronní komunikaci mezi klientem a serverem. Jelikož se jedná o novější řešení, může dojít ke komplikacím týkajícím se např. nedostatků jazyku JavaScript.

3 Návrh aplikace

Aplikace by měla umožňovat validaci jak na straně klienta, tak na straně serveru. Co nejvíce možných funkcí by se mělo kontrolovat už na straně klienta. Z toho důvodu bude nutné graficky navrhnout, jak o případných porušeních validačních pravidel uživatele informovat. Inspirovat se můžeme např. sociálními sítěmi, kde jsou podobné věci řešeny pomocí toolbaru pod vybranými vstupy. Nejvíce ideální bude uživatele informovat po každé akci, kterou na daném vstupu provede. Bude ale nutné zkoumat, zda toto řešení nebude časově náročné a nedojde-li k výraznému zpomalení webové stránky. Validační moduly by měly být navrhnuty tak, aby programátor mohl snadno určit, jaké validace mají být na jednotlivém vstupu použity. Validace na serveru by se měla provádět ideálně automaticky. Na závěr se musí každý modul otestovat.

3.1 Struktura Aplikace

Aplikace je rozdělená do následující adresářové struktury. Knihovna se nachází ve složce app a vše ostatní týkající se samotné aplikace v ostatních složkách z důvodů přehlednosti.



Obrázek 1 Adresářová struktura

1. app - všechny validační moduly a knihovny

2. app\DataSource - moduly potřebné pro komunikaci s databází a Socket.IO

3. css - kaskádové styly

4. js - scripty potřebné pro běh aplikace a klientské části a serveru

5. node_modules - zde jsou nainstalovány všechny potřebné externí knihovny včetně Node.js serveru

6. tests - soubory potřebné k testování

Jak již bylo řečeno, aplikace bude využívat nový standart ECMAScript 6, který umožňuje základní objektové programování. Aplikace je tedy rozdělena do různých tříd.

Kromě klasických tříd, které např. řeší vulgarismus nebo validaci, je tu ručně navržený Interface. Byl navržen z důvodu, že ECMAScript 6 ještě neobsahuje definici interfaců. Pro dodržení struktury tříd lze následujícím způsobem vytvořit vlastní interface, který pomůže budoucímu rozšiřování aplikace.

```
export default class IValidateComponent {
constructor() {
if (Object.getPrototypeOf(this) === IValidateComponent) {
throw new TypeError("Cannot construct Abstract instances directly");
        }
    }

validate() {
if (Object.getPrototypeOf(this) === IValidateComponent) {
throw new TypeError("Cannot construct Abstract instances directly");
        }
    }

check() {
if (Object.getPrototypeOf(this) === IValidateComponent) {
throw new TypeError("Cannot construct Abstract instances directly");
        }
    }
}
```

Většina těchto tříd často využívá stejné funkce, jako např. odstranění speciálních znaků atd. Proto jsou podobné funkce umístěny do statické třídy. Tuto třídu mohou vybrané moduly využívat.

3.2 Zvolení a návrh databáze

Aplikace řeší například detekci vulgarismů. Z toho důvodu je nutné zjistit, jaké jsou možnosti uložení databáze vulgárních slov. Hned v úvodu se nabízela možnost ukládat taková slova do textového souboru, ale to má mnoho nevýhod, jako např. náročnou implementaci a testování. Také se nabízelo řešení v podobě MySQL, ale v takovém případě by se zvyšovaly požadavky na server a programátora, který by musel aplikaci nastavit pro svůj MySQL server ručně. Další nevýhodou je, že by se špatně přenášela data, jako jsou např. slovníky vulgarismů, které by se zřejmě musely přenášet v podobě create scriptů. Tato varianta byla z těchto důvodů vyhodnocena jako nevhodná, avšak bude umožněno, aby si programátor mohl zvolit jakoukoliv databázi.

Při podrobnějším zkoumání se tedy ukázala jako nejlepší varianta databáze SQLite. Výhodou SQLite je, že se jedná o malou databázi, která plně postačí pro naše účely. Databáze se totiž bude přenášet jako součást aplikace. Jedná se pouze o jeden soubor, pro jehož zpracování bude nutná jenom JavaScriptová knihovna, která bude součástí této aplikace. Výsledná tabulka pro ukládání vulgarismů může vypadat následovně.

vulgarism_en
word varchar(50) PK

Obrázek 2 Ukázka databáze

Aplikace má základní implementaci pro manipulování se svými daty. Z důvodů univerzálnosti mají všechny funkce na vstupu dvě základní hodnoty, a to název tabulky a jazyku. Pro specifikování tabulky musíme vědět název tabulky a jazyk. Jelikož všechny tabulky budou obsahovat pouze databázi slov, tak je automaticky vytvořený atribut word, který je zároveň primárním klíčem, což zajistí, aby se každé slovo vyskytovalo pouze jednou. Na následujícím zdrojovém kódu můžeme vidět vytváření nových tabulek pomocí této aplikace.

```
createTable(dataType, language = "cz") {
var table = dataType + "_" + language;
var db= this.db;
db.serialize(function () {
db.run("CREATE TABLE "+table+" (word STRING)");
    });
}
```

4 Validace

4.1 Důvody validace

Uživatel se odjakživa snažil obejít předepsaná pravidla. Validační pravidla se obcházejí většinou dvěma způsoby.

Jednak nedokonalostí řešení, jelikož žádný problém nelze 100% vyřešit. Vezměme si například řešení, které bude detekovat, zda zadané slovo je vulgární. Takové slovo totiž nemusí být gramaticky totožné s porovnávaným vulgárním slovem, přitom člověku bude jasné, že se o vulgární slovo jedná.

Jako příklad se dá uvést slovo "průduch". Jeho detekce se dá obejít drobným zdvojením písmen "průdduch ". Jelikož se jedná o aplikaci zaměřenou na češtinu, tak se dá využít vlastnosti, že naprostá většina slov neobsahuje duplicitní písmena. Pokud ano, jejich odstranění nemá vliv na význam slova. Slovo "průdduch" bude tedy programově možné upravit pomocí regulárních výrazů na "průduch". Po takové úpravě už bude detekce úspěšná. Existuje ale celá řada deformací náročnějších na detekci, kterým se v této práci budeme věnovat.

Druhý problém při řešení validací je, že z důvodů výkonu je třeba provádět validaci už na straně klienta, protože je časově náročné čekat na odpověď serveru. Tento problém se násobí při pomalém připojení uživatele k internetu. Umístění kódu do klientské části ale způsobuje další problém. Takové validace bude nutné řešit pomocí JavaScriptu, který se celý stáhne ke klientovi do prohlížeče. Uživatel znalý tohoto jazyku tudíž může zdrojový kód pozměnit a zcela obejít validaci, proto je nutné ověřovat výsledky i na straně serveru.

Další útok, který je nutné kontrolovat, je vkládání různých nebezpečných vstupů uživatelem, jako v případě Cross-site-scriptingu. Proto musí probíhat kontrola i na podobné případy. Mimo jiné také můžeme chtít kontrolovat maximální délku textu nebo slov, podobnost dvou textů a spoustu jiných věcí.

4.2 Řešené validační problémy

Nyní je nutná implementace zadaných validačních problémů. Zde je jejich stručný seznam.

1. Detekce vulgarismů
2. Validace SEO [17]
3. Ověřování formátu (telefon, email)
4. Srovnání podobnosti dvou textů (Levenshtein distance)
5. Ověření, zda je po tečce mezera a velké písmeno
6. Zabezpečení proti útoku cross-site-scripting

Často využívané pomocné funkce

Následující funkce a mnoho dalších nacházejících se ve třídě HelpFunction jsou používany při impleplementaci validačních knihoven. Jsou vloženy v této třídě z důvodu častého používání.

1. Odstranění duplicitních písmen ve slově
2. Odstranění speciálních znaků
3. Rozdělení věty do slov
4. Nalezení slova ve slově
5. Odstranění mezer

4.3 Vulgarismy

Pro detekci vulgarismů využíváme dvě základní pomůcky. První je již zmíněná databáze cizích slov uložená pomocí SQLite [14]. Tou druhou je porovnávání dvou textů. To je prováděno pomocí algoritmu „Levenshtein distance“ [5] s obtížností *O(m*n)*.

Vstupem funkce je text z webového formuláře. Tento text je parsován do pole slov. Algoritmus zjistí kolik, znaků je třeba změnit, aby vstupní slovo bylo stejné jako porovnávané slovo (v našem případě vulgarismus získaný z databáze). Poté vezmeme výsledný počet znaků a spočteme procentuální podobnost těchto dvou slov. Na základě tohoto čísla usoudíme, zda dané slovo je vulgarismus, nebo ne. Takto porovnáváme celé pole slov s celou databází vulgarismů.

Pro názornější příklad se nyní vraťme k našemu testovacímu slovu "průduch". Ukážeme si, jak detekce probíhá a možné deformace, kterými se uživatel může pokusit zkomplikovat detekci:

Možné deformace: průduch, prrůduch, pruduch, p.r.u.d.u.c.h, pr-u-duuch, p ruduch

Detekce takových slov v textu je prováděna tak, že se text rozdělí pomocí mezer do slov a ze slov se odstraní všechny speciální znaky a česká diakritika. Dále je využito to, že česká slova neobsahují téměř žádná opakující se písmena po sobě. Tím se zbavíme případných záměrných oklamání kontroly. Uvažujeme-li, že do funkce přijde jako vstup slovo „—průůůduch“, výsledkem bude „pruduch“. Výsledné slovo se porovnává s celou databází vulgarismů pomocí algoritmu Levenshtein distance. Výsledkem této funkce bude číslo, ze kterého spočítáme procentuální hodnotu vůči hledanému slovu. Tato hodnota bude porovnána s hodnotou určitého parametru třídy, jehož hodnota vymezuje hranici, od které je slovo vyhodnoceno jako vulgarismus. Defaultně je toto číslo nastaveno na 85 %.

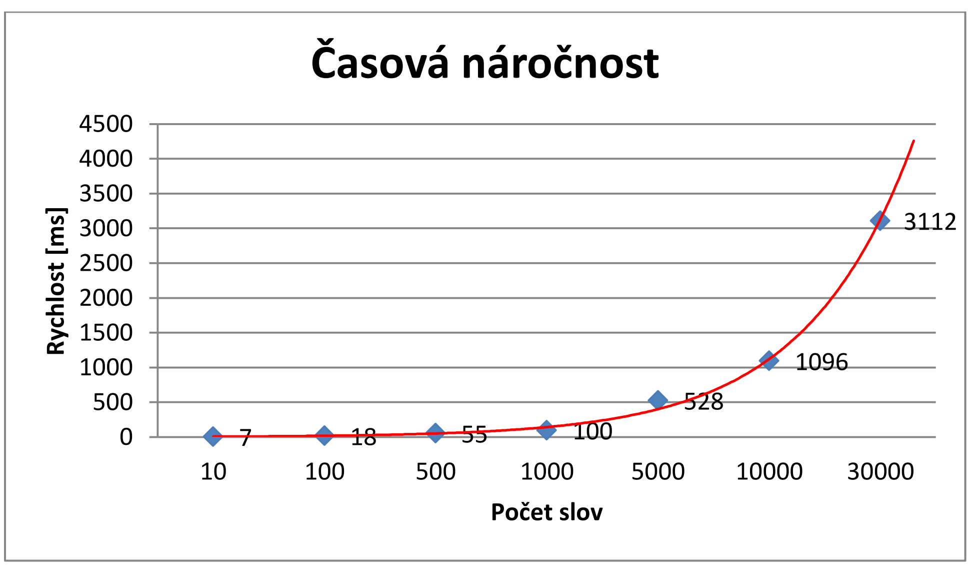
Toto řešení má nedostatky, když bude zadáno např. slovo „pru duch". Tento algoritmus provede to, že dané slovo rozdělí do dvou slov a vulgarismus nedetekuje. Z toho důvodu je tu záložní řešení, které spočívá v tom, že se text ověří ještě jednou a odstraní se z něj všechny mezery. Tím vznikne jeden velký řetězec všech slov bez mezer. Poté se berou všechny vulgarismy delší než 6 znaků a sleduje se, zda se vulgarismus v řetězci nevyskytuje. Pokud ano, je vyhodnoceno, že se v řetězci nachází vulgarismus. Příkladem takového vstupu může být "Ty jsi p růduch". Zde toto řešení úspěšné bude, protože slovo průduch se v řetězci "tyjsipruduch" nachází.

Jak už ale můžeme tušit, zde nastává problém. Může dojít k falešné detekci z důvodu, že kratší vulgarismy se můžou nacházet ve spojení dvou slov, a to s velkou pravděpodobností. Z toho důvodu se v řetězci hledají pouze vulgarismy delší než 6 znaků. Toto řešení je velice neohebné a ne příliš účinné, ale v kombinaci s prvním dosáhneme alespoň základní účinnosti.

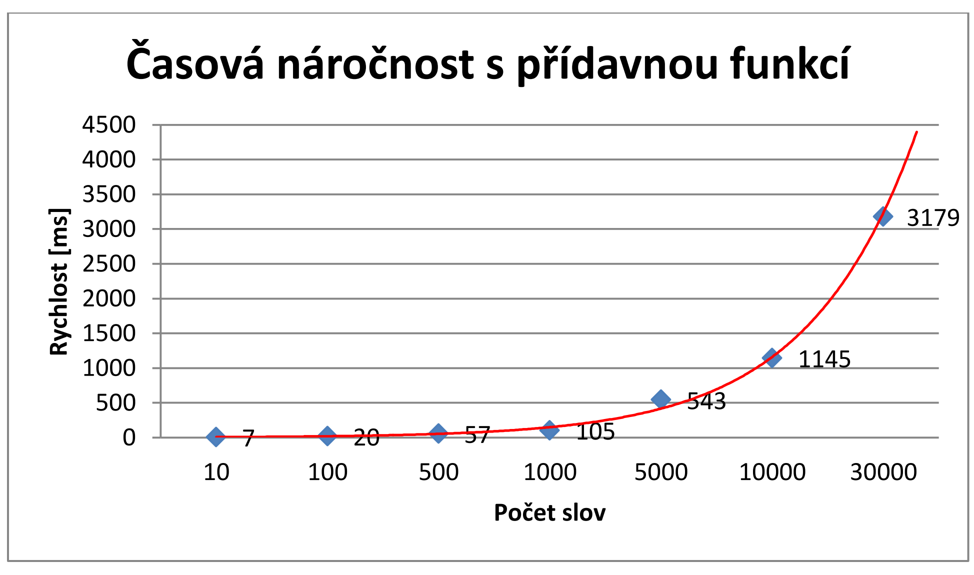
Z výše zmíněných důvodů je tento způsob detekce defaultně vypnut, ale jeho spuštění je možné zavoláním funkce stringInSubstringDetection. Pokud dojde k nalezení vulgarismu touto metodou, tak není možné provést automatickou opravu textu, ale můžeme uživatele upozornit na nevhodné vyjadřování.

4.3.1 Časová náročnost řešení

Další roli při detekci vulgarismů hraje časová náročnost. V následujících grafech můžeme vidět rychlost vykonání funkce pro detekci vulgarismů. První ukazuje rychlost vykonání detekce podle defaultního nastavení třídy. Druhá používá výše zmíněné záložní řešení, které spočívá v tom, že všechna slova spojí do jednoho velkého řetězce a hledá v něm vulgarismy delší než 6 znaků.



Obrázek 3 Časová náročnost při detekování vulgarismu



Obrázek 4 Časová náročnost detekce vulgarismu s použitím přídavné funkce

Postup měření

S využitím generátoru náhodného textu byly vytvořeny řetězce s počty slov: 10, 100, 500, 1 000, 5 000, 10 000, 30 000. Každý text zvlášť byl desetkrát prověřen třídou na detekci vulgarismů s defaultním nastavením a poté i s pomocnou funkcí. Z těchto deseti opakování byl spočten průměr, který byl zanesen do grafu. K měření rychlosti vykonání detekce byla použita knihovna qUnits.

Zhodnocení měření

Po zhodnocení naměřených dat je zjištěno, že do 1 000 slov se jedná o rychlost nepřesahující 100 ms, což lze na straně klienta tolerovat. K větší časové náročnosti dochází až při počtu okolo 5 000 slov, kdy je rychlost vykonání v průměru 550 ms. Taková délka už není příliš častá při zadávání textů do webového formuláře. Jednalo by se o vstup velikosti této bakalářské práce. Používat detekci při každé akci provedené na vstupu je doporučeno do počtu 1000 slov. Ověřování větších textů je doporučeno až při odesílání formuláře nebo pro detekci na straně serveru. Zároveň bylo vyhodnoceno, že přídavná funkce nemá vliv na výkon celé detekce.

4.3.2 Ověření funkčnosti

Detekce byla prověřena na úryvku první kapitoly knihy Boženy Němcové „Babička“. Detekce odhalila pouze vulgarismus ve slově „šukat”, který v daném díle nemá význam dnes používaného vulgárního slova. Došlo tedy pouze k úspěšné detekci. Knihovnu lze tedy na tomto dílčím testu označit jako funkční.

Dále byla detekce ověřena na úryvku knihy “Osudy dobrého vojáka Švejka” od Jaroslava Haška. Třída správně nalezla tři vulgarismy “hovno”, “blbá” a “blb”. Opět nedošlo k falešné detekci. Detekce však nenalezla slovo “blbý”. To je důsledkem vysoké nutné shody, která je nastavena na 85% slov. V takovém případě je nutné tuto hodnotu snížit a riskovat tak falešnou detekci, nebo toto slovo přidat do slovníku.

Poslední ověření detekce proběhlo na článku “Před 700 lety se narodil velký Evropan Karel IV.” [20], kde nebyl nalezen žádný vulgarismus.

4.3.3 Neuronové sítě

Daleko lepším a v současných technologických podmínkách nejperspektivnějším řešením je detekce pomocí „neuronových sítí“ [15]. Neuronová síť je využívaná pro tvorbu umělé inteligence, kde taková síť je schopná se sama učit. To si můžeme ukázat na příkladu s dítětem, které neví, jak vypadá pes a kočka. Dítěti dáme 5 fotografií a řekneme mu: ”Toto je pes a toto kočka”. Poté mu dáme šestou fotografií a zeptáme se ho, co je to za zvíře. Nedojde-li od nás k žádnému chytáku, dítě bude schopné rozpoznat, co je pes a co je kočka. Stejný princip je u neuronové sítě, která je dnes velice používaná na programování různých robotů a u aplikací, kde např. dochází k rozpoznávání prostředí na základě obrázků. Toto řešení se však ukázalo jako příliš komplikované a nad mé znalosti. Jeho možnou implementací se bude možné zabývat v diplomové práci.

4.3.4 Zhodnocení

Používané řešení lze využít pro základní ochranu vstupů, ale ne jako 100% účinnou ochranu. K větší účinnosti by byla nutná implementace pomocí neuronových sítí, která může být tématem navazující diplomové práce.

4.4 Univerzální třída pro validaci textového formuláře

Třída TextValidator obsahuje větší množství validačních funkcí. Je to z důvodu, aby méně náročné funkce byly dostupné z jedné třídy. Některé z validací umožnují indikaci i vyřešení daného problému. Jedna z takových funkcí je ověření toho, aby po tečce byla vždy mezera. Je umožněno, aby pouze varovala, že text nedodržuje toto pravidlo, ale je i umožněno, že funkce vrátí opravený text.

4.4.1 Kontrola délky textu

Tato kontrola se využívá pro libovolné vstupy, kdy je možné hlídat maximální délku textu. To se dá využít např. u vkládání příspěvků na webovou stránku, kdy chceme omezit velikost textu psanou uživatelem.

4.4.2 Kontrola délky slova

Téměř totožná kontrola zaměřená pouze na délku jednoho slova. Jelikož délka slov zřídka přesáhne určitou délku, lze ohlídat, aby uživatelé tuto délku neporušovali.

4.4.3 Kontrola a validace, zda se za tečkou nachází mezera a velké písmeno

Tato validace upravuje základní gramatiku stavby vět. Uvažujme, že ze vstupu přijde věta „venku je hezky.asi se půjdu projít“. Můžeme vidět, že po tečce chybí mezera a velké písmeno. Tato chyba je v dnešní době velice častá. Funkce umožňuje automatickou opravu této chyby. Náš požadovaný výstup v tomto případě je „Venku je hezky. Asi se půjdu projít“. Při validaci musíme dát ale pozor na zkratky. Přijde-li nám bez ošetření na vstup „Firma je s.r.o.“, neošetřená funkce by mohla vrátit výstup „Firma je s. R. O.“. Tato funkce může vracet dva výstupy.

1, Vrátíme pouze True nebo False a programátor s těmito informacemi naloží podle svého uvážení (např. upozorní uživatele pomocí toolbaru nebo ho upozorní při odeslání formuláře).

2, Druhá výstupní hodnota vrátí opravený text.

Řešení

Pro řešení byla použita databáze SQLite, kde je tabulka všech zkratek obsahujících tečku, které chceme zachovat. Funkce před provedením opravy zkontroluje, zda text neobsahuje nějakou zkratku z databáze. Pokud ano, třída místo této zkratky vytvoří placeholder (zástupné slovo). Toto slovo vznikne tak, že se z této zkratky odstraní tečky a vloží se mezi „%%“. Bude-li taková zkratka „s.r.o.“, placeholder bude vypadat následovně „%%s r o%%“. Takový řetězec už validace nijak nedeformuje. Nyní funkce může provést samotnou validaci pomocí regulárního výrazu. První část zjistí pomocí regulárního výrazu hledanou gramatickou chybu v textu a předá ji funkci, která ji opraví, tzn. přidá mezeru a zvětší první písmeno. Po opravení celého textu se placeholdery zamění zpátky za zkratky.

4.4.4 Zabezpečení před XSS

Zabezpečení proti útoku cross-site-scripting, který zabrání uživateli vkládat do vstupů vlastní scripty. K tomu je využita knihovna XSS [6], která dokáže odstranit všechnu nebezpečnou syntaxi:

Vstup

```
<script>alert(test);</script>
```

Výstup

```
alert(test);
```

4.4.5 Kontrola Emailu

Kontrola emailu probíhá pomocí "regulárního výrazu" [18]. Tato kontrola umožňuje zadat email obsahující subdoménu, jako např. test@email.subemail.com. Toto řešení neumožňuje kontrolovat naprosto všechny možné případy definované jako "validní email" [19]. Umožňuje však validaci většiny emailů používaných v dnešní době.

4.4.6 Kontrola telefonního čísla

Tato kontrola ověřuje, zda zadané telefonní číslo odpovídá předpsanému formátu. Tento formát se ale liší téměř v každém státu. Proto dalším vstupem funkce, umožňující tuto kontrolu, je výběr lokalizace. Lokalizace je dafaultně nastavena jako "cz", jejíž podpora je implementována v této aplikaci.

5 Využívané balíčky

5.1 NPM

Knihovna npm (Node package manager) [12] se používá pro instalování dalších knihoven do aplikace. Pokud vytváříme novou aplikaci, je nejdříve potřeba knihovnu pro projekt incializovat pomocí příkazu npm init. Po vykonání příkazu se v kořenovém adresáři vytvoří soubor package.json. V tomto souboru jsou poté zaznamenávány všechny knihovny nainstalované pomocí tohoto balíčku. K nainstalování nové knihovny stačí zadat např. npm install express. V takovém případě ale nedojde k záznamu o této knihovně do package.json. To musíme učinit ručně nebo zadáním parametru –safe. Pokud budeme chtít aplikaci přemístit na jiný server, tak nemusejí být tyto často objemné knihovny přemisťovány spolu s aplikací, protože díky souboru package.json máme záznam názvů a verzi všech knihoven, které musí být nainstalovány. K jejich opětovné instalaci stačí zadat příkaz npm install.

5.2 Babel

Babel [9] je JavaScriptový kompilátor pro ECMAScript. Je využíván pro kompilování standartu ECMAScript 6 do staršího a prohlížeči podporovaného ECMAScript 5. Tento kompilátor se běžně využívá pro kompilaci scriptů na serveru. Při implementaci serveru často využíváme celou řadu knihoven. Takové knihovny můžeme např. importovat pomocí následujícího příkazu:

```
var express = require('express');
```

Obdobný zápis pomocí ES6:

```
Import express from 'express';
```

Výše uvedené příkazy prohledají složku /node_modules a pokusí se v ní nalézt export pro express, pokud existuje. Tyto příkazy nebudou ale fungovat v prohlížeči, protože do prohlížeče musíme nahrávat scripty pomocí <script></script>. Tuto syntaxi ale nemůžeme využít, protože podobných souborů můžou být desítky až stovky a nemůžeme server zahlcovat tolika http requesty.

Takovou situaci tudíž musíme vyřešit ještě před používáním aplikace a odesílat uživateli jeden kompletní soubor obsahující všechny potřebné zkompilované scripty. K tomu můžeme využít Browserify.

5.3 Browserify

Browserify [10] řeší novinku, kterou zavedl Node.js, a to možnost modularizovat kód. Modulární systém potřebuje ke své funkcionalitě dvě věci, a to importovat a exportovat. Jak již bylo řečeno, problém nastává, pokud budeme chtít takovou syntaxi využít v prohlížeči. Musíme všechny potřebné knihovny odeslat uživateli v jednom kompletním souboru a to je to, co browserify umožnuje. Zkompiluje všechny importované knihovny, které jsou využívány.

Browserify ale neřeší zápis pomocí ES6

```
Import express from 'express';
```

Je tedy potřeba vše zkompilovat do ES5, ideálně pomocí Babelu. Knihovna, která umožnuje funkce Babelu i Browserify zároveň, má název Babelify [8]. Díky ní můžeme vše provést pomocí jednoho příkazu.

```
browserify script.js -o bundle.js -t [ babelify --presets [ es2015
react ] ]
```

5.4 Node-Inspector

Jedná se o utilitu na debugování JavaScriptu přímo v prohlížeči. Node-InspectorUmožňuje nám za běhu scriptu zasahovat do jeho obsahu. Dále je zde umožněno vkládat breakpointy a vykonávat scripty krok po kroku. Umožňuje nám vidět hodnoty proměnných a sledovat hardwarové zatížení procesoru. Usnadňuje také navigaci v souborech přímo v prohlížeči.

5.5 XSS

Knihovna XSS [5] umožňuje zabezpečení vstupů před útoky Cross-site scripting. Tento útok spočívá v tom, že útočník využívá bezpečnostních chyb webové aplikace a podstrčí tak do stránek vlastní JavaScriptový kód, čímž může narušit vzhled, funkčnost nebo získat případné citlivé údaje o uživatelích.

Pro ošetření vstupu před tímto útokem stačí zavolat funkci xss po importování knihovny xss.

```
var xss = require('xss');
var html = xss('<script>alert("xss");</script>');
```

5.6 SQLite 3

SQLite3 [13] je knihovna implementující funkce databázového systému SQLite. Před začátkem použití této knihovny ji musíme nejprve importovat a vytvořit instanci, kde jako parametr přijde databázový soubor.

```
Var sqlite3 = require("sqlite3").verbose();
var db = new sqlite3.Database('database.db');
```

Nyní máme vše připraveno pro používání. Vybrat data z databáze pomocí klasického selectu můžeme např. takto:

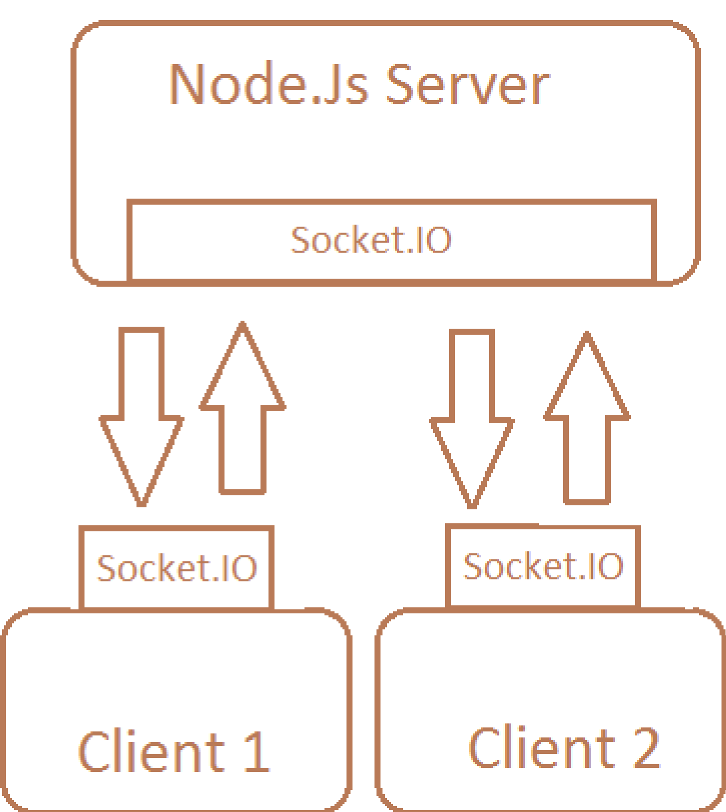
```
var ids = [];
db.serialize(function () {
db.each("SELECT id  FROM user", function (err, row) {
ids.push(row.id);
})
});
```

Kde each je cyklus pro vybrání všech požadovaných dat s výstupem row, což je objekt, který obsahuje hodnoty všech požadovaných atributů. Následně vše vložíme do našeho připraveného pole pomocí funkce push s parametrem row.id pro získání atributu id z databáze.

Za zmínku také stojí, že tento balíček podporuje i ochranu proti SQL injection. SQL injection je útok, který se pokouší přes neošetřené vstupy vkládat část SQL dotazů a narušit tak bezpečnost naší databáze. Jedna z nejúčinnějších obranných technik proti tomuto útoku jsou Prepared statments (předpřipravené dotazy). Fungují na principu, že napíšeme SQL dotaz, kde se místo proměnné, kterou chceme do dotazu vložit, napíše zástupný znak „?“. Proměnou vložíme až příkazem run, kde dojde k ošetření a odeslání dotazu pomocí dotazu finalize.

5.7 Socket.io

Balíček Socket.io [11] umožní klientské části asynchronně komunikovat se serverovou části. Socket.io otevře mezi serverem a prohlížečem obousměrný kanál, pomocí kterého mohou obě strany nezávisle na sobě spolu komunikovat. Socket.io funguje na bázi tzv. Socketů, což je virtuální spojení dvou procesů. Sockety [1] se používají pro meziprocesovou komunikaci. Z toho vyplývá, že můžeme napsat dva programy, které spolu mohou navzájem komunikovat. To můžeme vysvětlit i na příkladu, který je použit v této aplikaci. Jeden program bude server, a ten bude naslouchat na určitém portu a čekat na připojení druhého programu - klienta. Poté mohou obě strany vzájemně komunikovat.



Obrázek 5 Schéma popisující funkčnost Socket.Io

6 Vytvoření serveru

6.1 Stáhnutí dat

Vše potřebné máme nainstalované a nyní přejdeme k „vytvoření serveru“ [4]. Nejprve si připravíme data, která potřebují třídy pro svoji práci. V našem případě si stáhneme všechny vulgarismy. Při potřebě jiných dat můžeme zdrojový kód o ně rozšířit. Nejprve si však importujeme knihovnu fs, která má za úkol práci se soubory. My ji využijeme k načtení naší SQLite Databáze pomocí funkce existsSync(file). Tato funkce je synchronní verze exists, kterou musíme využít, protože potřebujeme vyčkat, než se databáze načte, jelikož ji nutně potřebujeme k následující práci. Nyní už můžeme používat třídu SQLiteDataSource, která má za úkol práci s databází. Výsledný zdrojový kód může vypadat následovně:

```
var fs = require("fs");
var file = "app/DataSource/database.db";
var exists = fs.existsSync(file);
```

Nejprve si importujeme třídu SQLiteDataSource. Na ní si demonstrujme nový způsob importování modulů. Nejprve napíšeme klíčové slovo IMPORT, po něm následuje parametr name. Parametr name je název objektu, který bude přijímat importované členy. Tyto členy se nacházejí v JavaScriptovém souboru daného modulu.

```
export default class SQLiteDataSource
```

Název SQLiteDataSource tedy použijeme v našem případě jako parametr name po klíčovém slovu IMPORT. Samozřejmě takový soubor může obsahovat více exportů. Při importování můžeme definovat, že chceme buď všechny komponenty souboru pomocí "*", nebo pouze vybrané vložením do "{}". Pro změnu definovaného parametru name můžeme použít Alias. Ten se definuje klíčovým slovem "as" a názvem požadovaného aliasu. Po parametru name následuje klíčové slovo FROM. Po něm je opět nutno vložit jako parametr cestu a název vybraného JavaScriptového souboru. Není nutné psát příponu ".js". Celou tuto syntaxi zatím nepodporuje žádný prohlížeč. Jejich používání vyžaduje kompilaci pomocí některého z kompilátorů, jako např. Babel, Traceur Compiler nebo Rollup. Celý import v našem případě bude vypadat takto:

```
import SQLiteDataSource from './app/DataSource/SQLiteDataSource';
```

Nyní máme připravený objekt, pomocí kterého můžeme vytvořit instanci pro manipulaci s tabulkou. Pro vytvoření instance potřebujeme jako parametr již připravený databázový soubor. Ten konstrutor předá knihovně pro obsluhu SQLite3, která umožní oboustrannou správu dat. Nyní si již můžeme říci o požadovaná data. V našem případě chceme všechny vulgarismy. Ty se nacházejí v tabulce vulgarism_cz. Jako parametr však stačí pouze vulgarism. Kvůli podpoře více jazyků můžeme zadat druhý parametr se zkratkou požadovaného jazyka. Standardně je však jazyk nastaven jako "cz", proto v našem případě nemusíme zadávat druhý parametr. Funkce nám jako výstup vrátí pole všech slov. Teď už nic nebrání vytvoření serveru, který bude zpracovávat POST požadavky a zároveň je validovat.

6.2 Připravení knihoven a modulů pro vytvoření serveru

Pro vytvoření serverového a klientského prostředí musíme nejprve definovat http interface [3]. K načtení potřebujeme klíčové slovo require, které budeme používat k načítání všech Node.js modulů. Nejprve musíme načíst třídy UrlParser a ValadionHandler, které jsem napsal pro účely této aplikace. UrlParser slouží k převedení POST požadavku, který přichází v URI formátu. Jako výstup vrací pole objektů, které obsahují parametry name, type a value. Parametr name v této aplikaci využívá možnost definovat type, který může vypadat následovně:

```
<textarea class="czechTools" name="user[email]">
```

To může ulehčit případnou validaci, protože nebudeme muset definovat na straně serveru, jaký výstup má být například validován jako email nebo telefonní číslo. Takto můžeme lehce poznat, jakým způsobem má být požadavek automaticky validován. Tuto automatickou validaci má nastarosti ValadionHandler. Zde může programátor definovat, co se má stát při splnění těchto požadavků. V této aplikaci pouze vypíšeme, že byly požadavky splněny nebo nesplněny, a vrátíme hodnotu true nebo false.

Velice důležitá věc, kterou potřebujeme, je konfigurační modul. Tento modul obsahuje údaje o IP adrese, doméně a portu. Tyto informace využijeme pro nastavení našeho serveru. Je nutné tento soubor změnit při nahrání aplikace na některý z hostingů.

```
var config = require('./config');
```

Poslední z modulů, který budeme potřebovat, je modul express. Hlavní věc, kterou tento modul umožňuje, je poskytnout prostředky pro HTTP server, což je skvělým řešením pro single-page aplikace nebo webové stránky. Pouze importovat knihovnu však nestačí, musíme ještě zavolat funkci express(), která vrátí potřebný objekt.

```
var express = require('express');
var app = express();
```

Nyní již máme vše připravené a můžeme přejít ke konfiguraci a implementaci serveru.

6.3 Vytvoření serveru pro naslouchání požadavků

Kdykoliv budeme potřebovat obsloužit požadavek z portů, budeme muset vytvořit objekt webového serveru. To se provádí pomocí funkce createServer. Potřebujeme k tomu náš připravený http modul.

```
var server = http.createServer (function (request, response) {
// Zde se nachází program
});
```

Funkce, která je předána funkci createServer jako parametr, je volána jednou pro každý http požadavek. Takže je to jakýsi správce požadavků. Výstupem této funkce je EventEmitter, pomocí kterého můžeme implementovat posluchač požadavků později.

```
var server = http.createServer ();
server.on ( "žádost", function (request, response) {
// Stejný program můžeme vložit sem
});
```

Když požadavek dorazí na server, daný uzel volá správce požadavků. Ten obsahuje řadu užitečných objektů, jako jsou url nebo method. Ty můžeme využít k rozeznání toho, z jaké url požadavek přišel nebo pomocí jaké metody byl poslán. Oba tyto způsoby jsou v této aplikaci využívány a můžeme je získat např. takto:

```
var method= request.method;
var url = request.url;
```

Nyní přejedeme k deklaraci EventEmittoru. Index.html obsahuje následující formulář:

```
<form method="post" action="/">
<textarea type="text" class="czechTools"
name="user[email]"></textarea>

<textarea type="text" class="czechTools"
name="user[email]"></textarea>

<textarea type="text" class="czechTools"
name="user[email]"></textarea>

</form>
```

Víme, že budeme zpracovávat metodu post. V aplikaci sdělíme, že pokud na server přijde požadavek POST, tak ať provede validaci všech přijatých hodnot z daného formuláře.

```
if (req.method == 'POST')
```

Nyní musíme získat všechna data, která touto metodou přišla na webový server. Ty získáme zavoláním funkce on s parametrem data a funkcí, která načte všechna data, která přišla tímto požadavkem.

```
var body = '';
req.on('data', function (data) {
body += data;
});
```

Výsledná data máme načtena v proměnné body. Tato data jsou prozatím v následujícím formátu:

```
Body: user%5Bemail%5D=eqw&user%5Bemail%5D=eqw&user%5Bemail%5D=eqwe
```

Na parsování těchto dat potřebujeme výše zmíněnou knihovnu urlPraser, která tato data rozparsuje do pole objektů, se kterým se nám bude lépe pracovat.

```
var array = urlParser.parseUrl(body);
```

Tato rozparsovaná data mohou být buď validována ručně, nebo můžeme k jejich validaci využít další funkce této aplikace s názvem validateInput. Ta vrací hodnoty true nebo false v závislosti na tom, jestli tato data splňují požadavky pro daný typ. Např. pokud objekt typu email nebude mít hodnotu, která je ve formátu emailu, tak funkce vrátí false. Pokud objekt neobsahuje žádný typ, tak funkce vrátí true. Provedeme cyklus, který projde celé pole objektů a provede validaci a kontrolu podle jejich typů.

```
for (var value of array) {
urlParser.validateInput(value,vulgarism);
}
```

Po konci zpracování tohoto požadavku chceme uživatele poslat např. na jinou stránku. K tomu využijeme knihovnu fs, která načte cílový html dokument. Dále musíme poslat hlavičku, která řekne, že po ní následující hodnota funkce end je typu html.

```
var html = fs.readFileSync('index.html');
res.writeHead(200, {'Content-Type': 'text/html'});
res.end(html);
```

Aby se nám všechny používané scripty a styly nahrály do prohlížeče je nutné je tam odeslat. To učiníme podobným způsobem jako s uvedeným nahráním souboru index.html. Jediný rozdíl je v tom, že pokud se prohlížeč snaží poslat požadavek o soubor s cestou /css/czechTools.css, tak mu zašleme tento soubor s hlavičkou pro kaskádové styly.

```
else if (req.url == '/css/czechTools.css') {
    var html = fs.readFileSync('css/czechTools.css');
    res.writeHead(200, {"Content-Type": "text/css"});
    res.end(html);
}
```

Na konci musíme nastavit, na jakém portu a adrese bude server naslouchat. K tomu slouží funkce Listen a data získána z konfiguračního souboru:

```
server.listen(config.port, config.domain, function () {
    console.log("Listening on " + config.domain + ", server_port " +
config.port)
});
```

6.4 Nastavení Socket.Io

Nyní už pouze musíme nastavit obousměrný kanál pro komunikaci klientské a serverové části. Nyní je potřeba importovat knihovnu socket.io s parametrem již vytvořeného serveru.

```
var io = require('socket.io')(server);
```

Jako poslední je potřeba odeslat pole vulgarismů, které budeme potřebovat pro validaci na straně klienta. K tomu se využívá funkce emit, pomocí které pošleme data v požadavku s názvem vulgarism a následně přijmeme ověření v požadavku checkConnection, abychom věděli, že vše proběhlo v pořádku.

```
io.on('connection', function (socket) {
    socket.emit('vulgarism', {words});
    socket.on('checkConnection', function (data) {
        console.log(data);
    });
});
```

7 Realizace klientské části

Po nastavení a spuštění našeho serveru je již přístupná stránka na adrese 127.0.0.1:8080. Jedná se o jednoduchou stránku, kde demonstrujeme vybrané validace, které následně budou odeslány na server. Je důležité říci, že se jedná o naprosto totožné zdrojové kódy, jaké jsou použity i na straně klienta. Nedochází tudíž k žádné duplicitě. Jelikož v nich ale používáme scripty, které obsahují syntaxi, které dnešní prohlížeče ještě nerozumí, tak musíme přegenerovat zdrojové kódy, které zde chceme používat pomocí babelify.

7.1 Vytvoření Toolbaru

Představme si textové pole, u kterého chceme, aby byla dodržena určitá pravidla. Například chceme, aby textové pole nesmělo obsahovat žádné vulgarismy a mělo určitou délku. Není potřeba jen detekovat vybrané problémy, ale informovat i uživatele, že došlo k jejich porušení. Ideálně by měl být informován po provedení jakékoliv akce. K těmto účelům byl implementován toolbar, který může obsahovat libovolné množství upozornění týkajících se validací.

Obrázek 6 Ukázka Toolbaru

Je navržen tak, aby bylo na programátorovi, zda chce panel pro daný vstup použít, a aby bylo zároveň snadné ho případně modifikovat. V této aplikaci se jeho implementace nachází v souboru czechTools.js. V našem příkladu detekujeme délku textu, délku slov a vulgarismy. Nyní si ukážeme, jak taková implementace může vypadat. Nejdříve si však řekneme, co je potřeba pro aktivaci tohoto panelu. Programátor musí daný vstup obalit do divu s třídou czechToolsItemDiv. Poté už je pouze nutné danému vstupu přiřadit třídu vybrané implementace. V tomto příkladu to bude třída czechTools. Nyní si řekneme, co musíme udělat pro implementaci takového JavaScriptového souboru.

Nejprve je nutné importovat všechny moduly, které budeme potřebovat pro naši práci. První budeme potřebovat knihovnu pro detekci vulgarismů a validaci textu.

```
import VulgarismDetector from '../app/VulgarismDetector;
import TextValidator from '../app/TextValidator;
```

Budeme požadovat, aby daná funkce reagovala při každé změně textového vstupu.

```
$(".czechTools").keyup(function ()
```

A budeme chtít, aby funkce reagovala pouze na naši třídu.

```
if ($(this).parent().is('div.czechToolsItemDiv'))
```

Vynulovala se při každé akci.

```
$(this).parent().find('.reportDiv').remove();
```

Poté už následuje pouze snadná implementace vulgarismů, kde zavoláme naši třídu a při úspěšné detekci i naši funkci insertStatus() s libovolným textem.

```
function isVulgarism(text,item) {
var test = new RemoveVulgarism(window.vulagrism);
if (test.isVulgarism(text.split(" "))) {
insertDiv(item, "", "Vulgarism detected")
    }
}
```

Funkce insertStatus vloží právě do divu czechToolsItemDiv další div, který obsahuje právě jeden status. Statusů ovšem může být neomezené množství. Případné úpravy designu se dají provést v souboru czechTools.css. Funkce umožňuje vložit libovolný text a případnou hodnotu.

```
function insertStatus(item, value, text) {
$(item).after("<div class='reportDiv'>" + text + value + "</div>");
}
```

7.2 Načtení dat přes Socket.IO

Na serveru je již nastaveno, že socket.io naslouchá na portu 8003. Už pouze stačí načíst data a vložit do proměnné, kterou bude používat funkce pro detekci vulgarismů.

```
var socket = io.connect('http://localhost:8003');
socket.on('vulgarism', function (data) {
    window.vulagrism = data.words;
    socket.emit('checkConnection', 'Vulgarism received');
});
```

Po úspěšném přijetí dat odešleme na server potvrzení o přijetí, ať víme, že aplikace funguje správně. Pokud bude akce úspěšná, můžeme v konzoli vidět hlášku "Vulgarism receive", kterou jsme poslali na server funkcí socket.emit v požadavku s názvem "checkConnection".

8 Používání aplikace

8.1 Klientská část

Nyní je již aplikace hotová a můžeme ji spustit a ukázat si její funkčnost. Zavoláním příkazu `babel-node server.js` nebo `npm start` nás odkáže konzole na webovou stránku na adrese http://127.0.0.1:8080. Zde již máme náš připravený index.html, kde jsou importované všechny potřebné knihovny včetně všech zkompilovaných modulů pomocí babelify. Na stránce máme požadovaný formulář, kde by měly být funkční validace pro délku slov a textu, validace telefonního čísla a emailu a hledání vulgarismů

Obrázek 7 Ukázka úspěšné detekce webové stránky

Můžeme vidět, že naše vulgární slovo bylo nalezeno. Můžeme ale také vidět příklady dalších validací, které knihovna umožnuje, jako například validaci telefonního čísla a emailu. Text v toolbaru je samozřejmě možné měnit.

Na následujícím obrázku můžeme také vidět, že je možné provádět detekci opačně, tzn. upozornit uživatele, pouze pokud požadavek splní.

Test knihovny

Emailová adresa:

test@centrum.cz

Emailová adresa zadána

Telefonní číslo:

774-129-096

Telefonní číslo zadáno

Popis:

Test textu !

Délka textu: 12

Odeslat

Obrázek 8 Ukázka možné alternativní detekce

8.2 Server

Nyní se podívejme, v jaké formě přijde požadavek na server a jak s ním můžeme pracovat. Jelikož použité textové pole bylo typu text, tak je daný vstup defaultně validován pouze na vulgarismus a XSS útok.

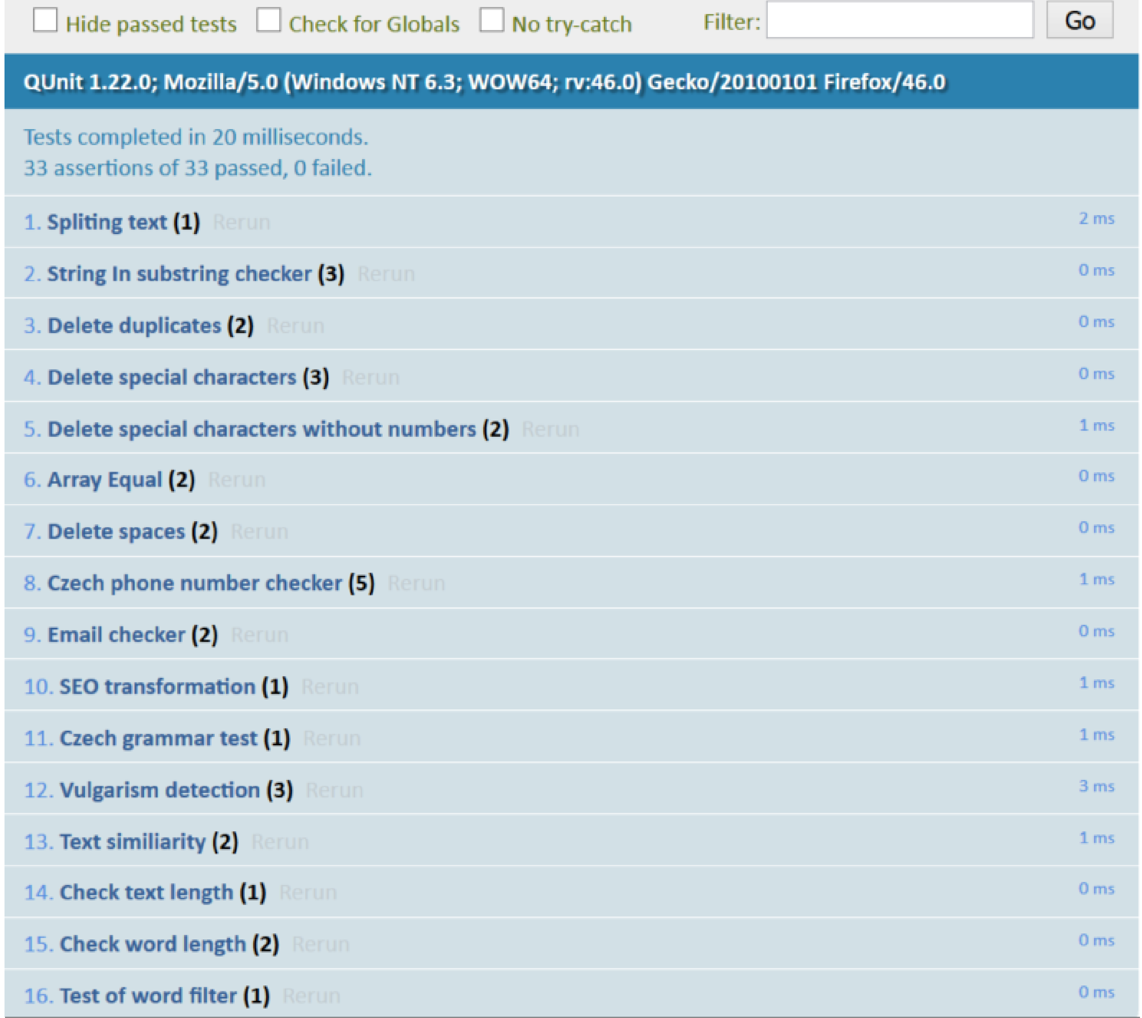
```
POST
Body: user%5Btext%5D=Ty+jsi+pr%C5%AF%C5%AF%C5%AFdu.ch+%21
{ name: 'user', type: 'text', value: 'Ty jsi průůůdu.ch !' }
true
```

Obrázek 9 Ukázka konzolového výstupu

Na obrázku můžeme vidět data, která nám přijdou, a objekt, do kterého je vkládáme. Jak bylo řečeno dříve, můžeme vidět název vstupu i jeho typ. Pro nás je nejdůležitější jeho typ. Dále můžeme vidět, že i na serveru byl správně detekován vulgarismus. Tím pádem tedy máme bezpečně zajištěno, že v textu nebude žádný knihovnou detekovatelný vulgarismus a daný text můžeme např. vložit do databáze.

9 Testování

Každá použitá funkce je otestovaná pomocí JavaScriptové knihovny qUnits [7]. Ta umožňuje klasické unit testování, jehož principem je automatické testování každé elementární části kódu. V oblasti procedurálního programování tím můžeme chápat každou funkci. V objektovém programování každou třídu či metodu.



Obrázek 10 Ukázka výstupu qUnits při testování

Použití knihovny je snadné. Stačí použít html šablonu poskytovanou vývojářem, kde jsou nejdůležitější následující dva importované soubory.

```
<link rel="stylesheet" href="https://code.jquery.com/qunit/qunit-
1.22.0.css">
<script src="//code.jquery.com/qunit/qunit-1.22.0.js"></script>
```

Pak už si stačí vytvořit vlastní JavaScriptový soubor, kde už můžeme používat klasickou syntaxi pro unit testy. Pro použití musíme zavolat funkci QUnit.test a v ní použít funkci assert.ok pro kontrolu výsledku.

```
QUnit.test( "String In Substring", function( assert ) {
var str = "Hello world, welcome to the universe.";
var output = TextValidator.stringInSubString(str,"welcome")
assert.ok( output == "13", "Passed!" );
});
```

Jelikož musíme testovat opět naše soubory, které jsou napsány v ES6 a jsou modulární, tak musíme soubory zkompilovat pomocí babelify.

Závěr

Bakalářská práce se zabývala implementací knihovny pro validaci vstupních polí webového formuláře, kontrolujících se na straně klienta i serveru, a zkoumáním vhodných technologií, které jsou pro tento účel nejvhodnější.

Z důvodu nutné kontroly vstupů už na straně klienta bylo řešení pomocí PHP a Ajaxu vyhodnoceno jako nevhodné. Bylo zjištěno, že řešení by vedlo k duplicitě kódu a v případě častých validací i k výkonové zátěži z důvodu časté komunikace se serverem.

Pro řešení byl upřednostněn Node.js server s celou řadou přídavných knihoven. Díky tomu je umožněno programování klientské části i serveru pomocí JavaScriptu. To také minimalizovalo nutnost komunikování klientské části s částí serverovou.

V knihovně je zajištěna detekce vulgarismů, SEO validace, zabezpečení vstupů před útoky XSS, validace emailu a telefonního čísla, detekce a oprava vybraných gramatických pravidel. O výsledcích validací může uživatele informovat toolbar, který může být aktivován pod každým vstupem.

Byla zajištěna modulárnost na straně klienta i serveru pomocí nové verze JavaScriptu. Každá používaná metoda byla řádně otestována pomocí unit testů.

Knihovna je v současné době využívaná pro komerční účely a v budoucnu bude neustále zdokonalována. Řešení vulgarismů pomocí neuronových sítí může být předmětem případného pokračování v diplomové práci.

Seznam použité literatury

[1] VÁCLAVÍK, Jiří. Perl (74) - Sockety. *Linuxsoft* [online]. 2008 [cit. 2016-04-30]. Dostupné z: http://www.linuxsoft.cz/article.php?id_article=1571

[2] ZAKAS, Nicholas C. *Understanding Ecmascript 6* [online]. 2016 [cit. 2016-04-15]. 978-1593277574

[3] Anatomy of an HTTP Transaction. Nodejs [online]. [cit. 2016-04-30]. Dostupné z: https://nodejs.org/en/docs/guides/anatomy-of-an-http-transaction/

[4] NEŠETŘIL, Jakub. JavaScript na serveru: Začínáme s Node.js. Zdrojak [online]. 2010 [cit. 2016-03-26]. Dostupné z: https://www.zdrojak.cz/clanky/javascript-na-serveru-zaciname-s-node-js/

[5] The Levenshtein-Algorithm. *Levenshtein* [online]. [cit. 2016-05-11]. Dostupné z: http://www.levenshtein.net/index.html

[6] *Jx-xss* [online]. [cit. 2016-05-11]. Dostupné z: https://github.com/leizongmin/js-xss

[7] QUnit. *QUnit* [online]. [cit. 2016-05-11]. Dostupné z: https://qunitjs.com/

[8] *Babelify*[online].[cit.2016-05-11]. Dostupné z: https://github.com/babel/babelify

[9] Babel. *Babeljs* [online]. [cit. 2016-05-11]. Dostupné z: https://babeljs.io/

[10] Browserify. *Browserify* [online]. [cit. 2016-05-11]. Dostupné z: http://browserify.org/

[11] Socket.IO. *Socket.IO* [online]. [cit. 2016-05-11]. Dostupné z: http://socket.io

[12] NPM. Npmjs [online].[cit. 2016-05-12]. Dostupné z: https://www.npmjs.com/npm/open-source

[13] Node-sqlite3. *Github* [online]. [cit. 2016-05-12]. Dostupné z: https://github.com/mapbox/node-sqlite3

[14] SQLite. Sqlite [online]. [cit. 2016-05-12]. Dostupné z: https://www.sqlite.org/about.html

[15] Neural Network. *Neuralnetworksanddeeplearning* [online]. [cit. 2016-05-12]. Dostupné z: http://neuralnetworksanddeeplearning.com/

[16] DoS & DDoS. Incapsula [online]. [cit. 2016-05-12]. Dostupné z: https://www.incapsula.com/ddos/ddos-attacks/denial-of-service.html

[17] SEO. Whatisseo [online]. [cit. 2016-05-12]. Dostupné z: http://www.whatisseo.com/

[18] Email Validation. Stackoverflow [online]. [cit. 2016-05-15]. Dostupné z: http://stackoverflow.com/questions/201323/what-is-the-best-regular-expression-for-validating-email-addresses/201378#201378

[19] Email address. *Wikipedia* [online]. [cit. 2016-05-15]. Dostupné z: https://en.wikipedia.org/wiki/Email_address

[20] Před 700 lety se narodil velký Evropan Karel IV. *Novinky* [online]. [cit. 2016-05-14]. Dostupné z: http://www.novinky.cz/veda-skoly/403313-pred-700-lety-se-narodil-velky-evropan-karel-iv.html?sznhp%3Drss