
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**
**BRNO UNIVERSITY OF TECHNOLOGY**

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**
**FACULTY OF INFORMATION TECHNOLOGY**

**ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ**
**DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA**


# WEBOVÁ APLIKACE PRO INTUITIVNÍ SESTAVENÍ FILTRŮ TEXTU

**WEB APPLICATION FOR INTUITIVE COMPOSITION OF TEXT FILTERS**

**BAKALÁŘSKÁ PRÁCE**
**BACHELOR'S THESIS**


**AUTOR PRÁCE** **JAKUB SADÍLEK**
**AUTHOR**

**VEDOUCÍ PRÁCE** **prof. Ing. ADAM HEROUT, Ph.D.**
**SUPERVISOR**



**BRNO 2020**

**Vysoké učení technické v Brně**
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2019/2020

# Zadání bakalářské práce


Student: **Sadílek Jakub**
Program: Informační technologie
Název: **Webová aplikace pro intuitivní sestavení filtrů textu**
**Web Application for Intuitive Composition of Text Filters**
Kategorie: Uživatelská rozhraní

Zadání:

1. Prostudujte problematiku filtrace a zpracování textových souborů.
2. Sestavte dostatečně rozsáhlou a reprezentativní datovou sadu textových souborů a úkolů na jejich filtraci.
3. Navrhněte webovou aplikaci pro uživatelsky snadné prototypování a ladění filtrů pro zpracování textových souborů.
4. Implementujte navrženou aplikaci.
5. Testujte navrženou aplikaci na vhodných datech a s vhodnou skupinou testerů. Iterativně vylepšujte uživatelské rozhraní i funkčnost aplikace.
6. Demonstrujte použití aplikace na vhodné sadě dat.
7. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Ondřej Žára: JavaScript - Programátorské techniky a webové technologie, Computer Press, 2015
- Jon Duckett: Web Design with HTML, CSS, JavaScript and jQuery Set, John Wiley & Sons Inc., 2014
- Eve Porcello, Alex Banks: Learning React: Functional Web Development with React and Redux, O'Reilly Media, 2017

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 4, značné rozpracování bodu 5.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Herout Adam, prof. Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 1. listopadu 2019

# Abstrakt

Cílem této práce je poskytnout intuitivní a snadno použitelný nástroj pro pokročilou filtraci textu s možností jednoduchého prototypování a ladění, bez nutnosti znát techniky programování. Základním principem je volba textových nástrojů a jejich řazení do řady tzv. pipeline, což je typické pro shell, ze kterého aplikace čerpá inspiraci. Nástroje také lze dodatečně editovat či zaměňovat. Aplikace míří zejména na uživatele neznající tento princip nebo programátory, pro které je časově výhodné si nechat text takto upravit a následně si vygenerovat ekvivalentní shell výraz. Další způsob ladění je realizován pomocí tzv. breakpointů, přes které se lze snadno a rychle zaměřit na vybrané řádky textu. Aplikace tak nabízí běh ve dvou režimech, mezi kterými lze libovolně přepínat podle potřeb uživatele.



# Abstract

The aim of this thesis is to provide intuitive and easy to use tool for advanced text filtration with the option of easy prototyping and tuning, without the necessity to know programming techniques. The basic principle is the choice of text tools and their inserting into the sequence, so called pipeline, which is typical for shell, from which the application draws inspiration. Tools can be also additionally edited or swapped. The application is aimed primarily at users unfamiliar with this principle or at programmers, for whom it is time-efficient to have their text modified this way and afterwards generate equivalent shell expression. Another way of tuning is realized using so called breakpoints, through which it is easy to quickly focus on chosen lines of the text. This way, the application offers functionality in two separated modes, between which the users can switch anytime according to their needs.



# Klíčová slova

webová aplikace, filtrace textu, zpracování textu, uživatelské rozhraní, shell, react, material-ui, ace

# Keywords

web application, text filtering, text processing, user interface, shell, react, material-ui, ace

# Citace

SADÍLEK, Jakub. *Webová aplikace pro intuitivní sestavení filtrů textu.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, Ph.D.

# Webová aplikace pro intuitivní sestavení filtrů textu

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana prof. Ing. Adama Herouta, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Jakub Sadílek
27. května 2020


## Poděkování

Chtěl bych poděkovat svému vedoucímu práce prof. Ing. Adamovi Heroutovi, Ph.D. za cenné rady a originální nápady, které vedly moji práci správným směrem. Dále bych rád poděkoval všem, kdo se podíleli na testování za poskytnutí zpětné vazby.

# Obsah


**1 Úvod** **3**

**2 Zpracování textových dat a současný stav** **4**
2.1 Problematika zpracování textových souborů . . . . . 4
2.2 Řešení v dnešní době . . . . . 4

**3 Návrh webové aplikace** **7**
3.1 Požadavky na webovou aplikaci . . . . . 7
3.2 Uživatelské rozhraní . . . . . 9
3.3 Výběr nástrojů . . . . . 11
3.4 Cílová skupina . . . . . 12

**4 Použité technologie** **13**
4.1 Node.js . . . . . 13
4.2 React . . . . . 14
4.3 Material-UI . . . . . 20
4.4 Ace editor . . . . . 21
4.5 Web worker . . . . . 22
4.6 Selenium . . . . . 22
4.7 GitHub Pages . . . . . 24

**5 Implementace webové aplikace** **25**
5.1 Struktura projektu . . . . . 25
5.2 Dekompozice aplikace . . . . . 27
5.3 Proces zpracování textu . . . . . 29
5.4 Tvorba automatizovaných testů . . . . . 32
5.5 Nasazení aplikace . . . . . 33
5.6 Výsledná podoba aplikace . . . . . 34

**6 Sběr dat a testování** **37**
6.1 Testovací datová sada . . . . . 37
6.2 Automatizované testování . . . . . 39
6.3 Uživatelské testování . . . . . 40

**7 Závěr** **43**

**Literatura** **44**

**A Plakát** **46**

**B Obsah přiloženého paměťového média** **47**

**C Instalační manuál** **48**
C.1 Instalace aplikace . . . . . 48
C.2 Instalace testů . . . . . 49

# Kapitola 1

# Úvod

Náplní této práce je zpřístupnit nástroj pro pokročilou filtraci textu běžným uživatelům. Filtraci textových dat lze řešit v IT oblasti různými nástroji. Ovšem lidé neznalí programování tuto situaci musí řešit komplikovaným způsobem nebo využít již dostupná řešení, která jim alespoň zčásti pomohou. Větší problém pak nastává při složitější transformaci, kterou nelze vyřešit elementární operací.

V současné době existují na internetu různé nástroje řešící tento problém, které využije i znalý programátor kvůli jejich snadné dostupnosti, ale poskytují jen jednu operaci najednou, tudíž je potřeba text neustále kopírovat nebo v horším případě hledat jiné stránky poskytující právě požadovanou operaci. Na otázku „Jak se tento záznam na onom řádku promítl do výsledku?“ se odpověď hledá jen těžko. Pokud by někdy v budoucnu bylo potřeba provést transformaci znovu, je nutné celý proces opakovat a to stojí čas.

Hlavní motivací této práce je eliminovat výše uvedené nedostatky a poskytnout uživatelům komplexní nástroj s velkou variabilitou, ve kterém mohou snadno a efektivně sestavit výslednou podobu textu s možností prototypování a ladění. Nabídnout uživatelům aplikaci, ve které mohou odfiltrovat nepodstatné informace například z textových tabulek, zpracovat log soubory, a to vše naráz z jednoho místa.

V následujícím textu je podrobněji obsažena většina aspektů, které bylo potřeba při vývoji řešit. Přesněji v kapitole 2 je zmíněna problematika filtrování, proč má smysl data filtrovat a jsou zde představeny existující nástroje. Kapitola 3 obsahuje proces návrhu, zpracování požadavků na aplikaci, kritéria pro výběr nástrojů a nakonec je zmíněna cílová skupina uživatelů. Kapitola 4 se zaměřuje na nejpodstatnější technologie použité v aplikaci a hlouběji je popsán jejich princip a funkčnost. V kapitole 5 o implementaci je popsána dekompozice a struktura projektu, jak probíhá proces zpracování a ukazuje zajímavé části kódu. Závěr této kapitoly obsahuje ukázky výsledné aplikace. Poslední kapitola 6 se věnuje sběru testovacích dat, automatizovanému a uživatelskému testování včetně vyhodnocení.

# Kapitola 2

# Zpracování textových dat a současný stav

První část této kapitoly uvede čtenáře do problematiky zpracování textových souborů. Jsou zde popsány situace, kde se s touto problematikou setkáváme a jak vzniká. Druhá část kapitoly je věnována dostupným řešením, které lze v dnešní době použít včetně dostupných aplikací na internetu.

## 2.1 Problematika zpracování textových souborů

Řešit tento problém v IT průmyslu je běžnou praxí. Správci sítě se s ním potýkají například při analýze přístupových logů, systémoví administrátoři v operačních systémech nebo programátoři, jejichž programy na těchto datech závisí. Běžní uživatelé se s takovými daty mohou setkat při jednoduše překopírovaných tabulkách, v publikovaných seznamech nebo soupiskách či jinde. Také existují portály, které zveřejňují zajímavé datové sady z oblasti výzkumů jako například Kaggle[1]. Typicky jsou tato data rozsáhlá a jejich zpracování bez použití automatizace je časově náročné.

Odhadem při 300 a více záznamech nastává situace vhodná k použití automatizovaných nástrojů. Obzvlášť pokud se jedná o neseřazená data. Nejčastější operací bývá vyhledávání, které lze snadno provést klávesovou zkratkou „ctrl + F“ implementovanou ve většině prohlížečích. Pokud je potřeba data uložit, přeposlat nebo zveřejnit je vhodné odstranit nepodstatné informace a tím text modifikovat. Odstranění záznamu lze provést snadno odstraněním příslušného řádku v případě malého počtu, ale při odstranění, modifikaci informace v záznamu *(tj. sloupce)* nebo řazení nastává situace komplikovanější, jelikož musíme upravit každý jednotlivý záznam zvlášť.

## 2.2 Řešení v dnešní době

V dnešní době existuje celá škála možností jak z textu vyfiltrovat podstatné informace. Jedním ze způsobů je vytvořit si jednoduchý program nebo skript. Programátor vytvoří algoritmus, který realizuje transformaci a ten naprogramuje v libovolném jazyku. Dnes se může jednat o Python, PHP, Bash, JavaScript a mnoho dalších. Nevýhoda tohoto způsobu je, že čas strávený při implementaci může být zbytečně dlouhý a silně závisí na dovednostech

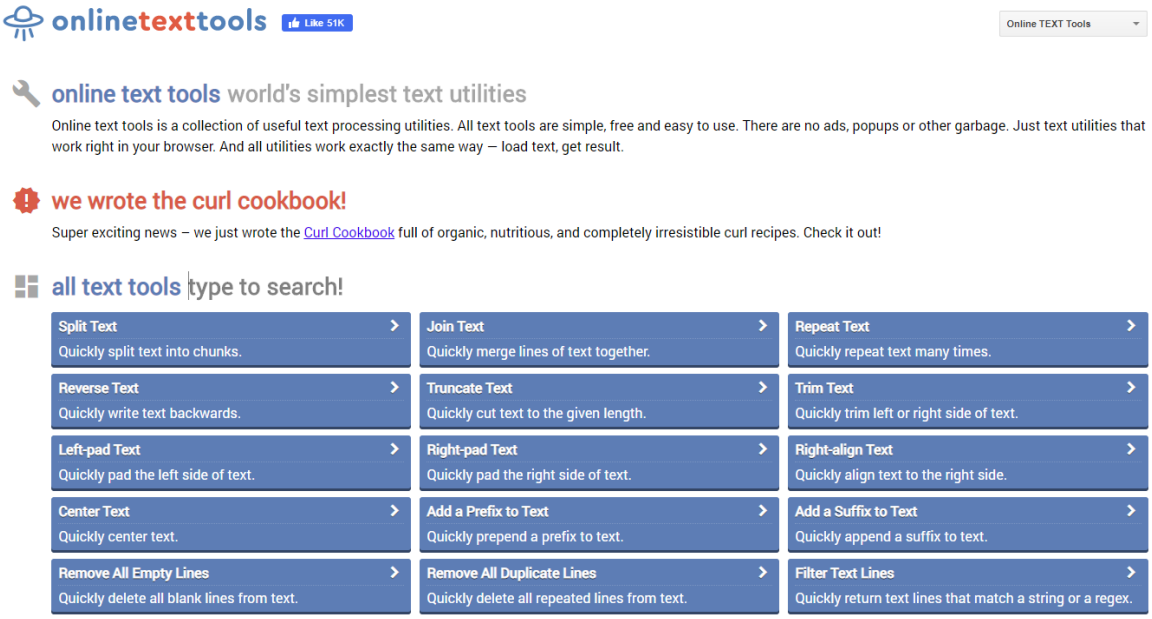
[1] Kaggle: https://www.kaggle.com

dané osoby, obzvlášť pokud jde o jednorázovou filtraci a program pak zůstane nevyužitý. V opačném případě je toto řešení velmi efektivní.

Co když jedinec není programátor? Pak nejspíše využije některých z nástrojů, které jsou zdarma snadno dostupné, nejspíše na webu. To je druhý způsob, kterým lze problém řešit. Tyto nástroje fungují na podobném principu: vložit vstupní text, provést operaci a vyjmout výsledek. Takových nástrojů existuje na internetu celá řada, následně si uvedeme příklad těch nejzajímavějších.

### Online text tools

Online text tools[2] je populární webová stránka pro zpracování textu, která nabízí velké množství nástrojů a v budoucnu plánuje přidat ještě další. Nabízí také vyhledávač, který umožňuje přehledně redukovat výběr nástrojů. Část stránky je znázorněna na obrázku 2.1. Jak je na samotné stránce uvedeno, funguje na principu „načíst text, získat výsledek“.



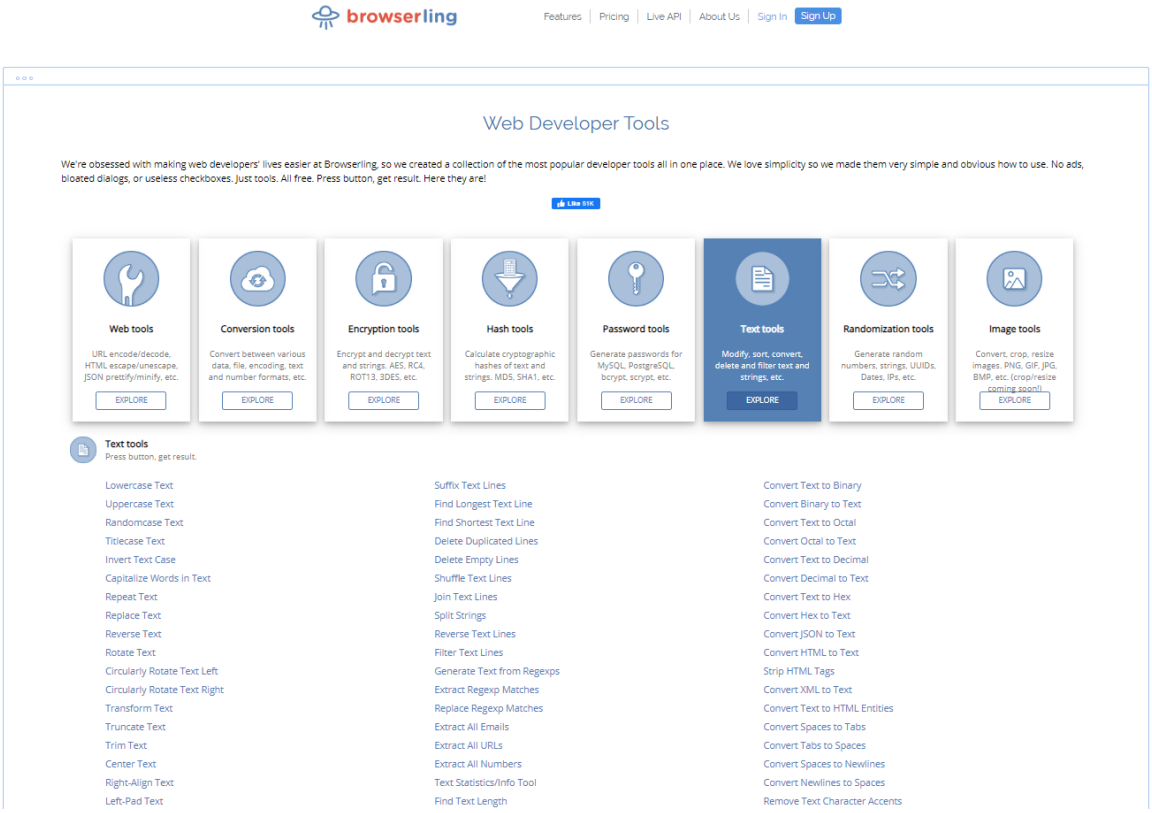

Obrázek 2.1: Ukázka úseku hlavní stránky aplikace online text tools, na které je zobrazena část nabízených nástrojů.

### Browserling

Browserling[3] je další zajímavý webový portál, který nabízí spoustu nástrojů, které zejména ocení vývojáři webových stránek. V nabídce jsou nástroje pro práci s obrázky, časem, šifrováním a další. My se zaměříme pouze na ty textové.

Celkem je obsaženo 113 textových funkcí, které lze použít, ovšem většina je velmi specifická, a tak mají jen velmi úzké spektrum využití jako například převody mezi kódováním nebo datovými formáty. Nástroje fungují také na principu vstup, jednoduchá operace a výstup. Přes velkou kolekci nástrojů si tu každý najde svoji podmnožinu, která se mu hodí. Náhled úseku stránky je zobrazen na obrázku 2.2.

[2]Online text tools: https://onlinetexttools.com
[3]Browserling: https://www.browserling.com/tools




Obrázek 2.2: Ukázka části galerie s textovými nástroji webové aplikace Browserling.

### Zhodnocení existujících řešení

V existujících testovaných aplikacích lze provést pouze jednu operaci v daný okamžik a při dalším zpracování je nutné hledat potřebnou funkci jinde. Některé aplikace neumožňují nahrát vstupní text ze souboru či stáhnout výsledek. Velikost vstupních elementů bývá malá a při obsáhlejších datech způsobuje nepřehlednost a ztěžuje orientaci v textu, zároveň aplikace neposkytují podpůrné funkce, které usnadňují dodatečnou manipulaci se vstupním textem. Většinou také galerie nebývají nijak členěny a nástroje jsou zobrazeny pouze v jednoduchém seznamu. Možnost prototypování nenabízela žádná z testovaných aplikací, a když si ke konci filtrace uživatel uvědomí, že ve výsledku opomněl některé záznamy, musí celý proces provést znovu.

U výše testovaných aplikací je velkou výhodu, že nabízí velkou rozmanitost nástrojů, proto hledání jiných služeb realizující požadovanou operaci nastává výjimečně a obsah těchto galerií byl inspirací při návrhu implementovaného řešení.

# Kapitola 3

# Návrh webové aplikace

Tato kapitola obsahuje celý proces návrhu aplikace. Jsou zde zmíněny původní záměry a řešení, které poskytuje a rozebrán vývoj uživatelského rozhraní. Další část kapitoly je zaměřena na výběr nástrojů, které byly v aplikaci implementovány. Na závěr je uvedeno na koho aplikace cílí a kdo by mohl být typickým uživatelem.

## 3.1 Požadavky na webovou aplikaci

Hlavním požadavkem na aplikaci je nabídnout uživateli nástroj, ve kterém lze snadno a efektivně vyfiltrovat podstatné informace z textových dat. Taková filtrace může obsahovat složitou transformaci, kterou je možné sestavit ze sekvence jednodušších, elementárních operací. Proces zpracování je pak možné realizovat pomocí zřetězeného zpracování (pipelining), kde výstup operace poskytne vstup následující operaci. Mezi další požadavky patří:

### Komfortní editor

Použity jsou celkem dva editory, jeden je určen pro vstup, druhý zobrazuje výsledek a je určen pouze ke čtení. Editor je základ celé aplikace, proto by měl nabízet také mnoho funkcí, které uživateli usnadní práci. Zejména se může jednat o práci se soubory, které ušetří některé nadbytečné úkony nebo o hromadné akce, které by musel uživatel jednotlivě a opakovaně provádět. S větším počtem uživatelů může také vzniknout více požadavků na operace se vstupními daty nebo jejich zobrazení. Implementované funkce do editoru tedy jsou:

- Manipulace se soubory
- Zpětné a dopředné vracení změn
- Zalamování textu
- Ovládání režimu
- Hromadné ovládání použitých nástrojů
- Vyhledávač v textu včetně regulárních výrazů

### Intuitivní a jednoduché rozhraní

Při tvorbě takového rozhraní je zapotřebí spousta testování, protože aplikace je určena pro široké spektrum uživatelů, musí rozhraní vyhovovat všem. Typicky náročnost použití rozhodne o tom, jak bude výsledek oblíbený. Uživatel musí intuitivně tušit kam se podívat a co kde hledat při jeho úkonech. Více o návrhu uživatelského rozhraní lze nalézt v následující kapitole 3.2, výsledná podoba je pak znázorněna v kapitole 5.6.

### Eliminace zbytečných úkonů

Tuto vlastnost můžeme pozorovat například při zpracovávání textu pomocí shellu, ze kterého byla převzata inspirace v mnoha směrech. Prakticky může stačit pouze jeden výraz k dosáhnutí výsledné podoby textu. Takto lze ušetřit zbytečné operace jako je překopírování, přesměrování stránek a nabídnout vše na jednom místě. Tento způsob je poměrně originální a žádná testovaná aplikace tuto možnost nenabízela.

### Prototypování a ladění

„Moment, výběr klasickým výrazem nestačí, ale musím použít regulární výraz!“ Nástroj lze jednoduše smazat nebo deaktivovat a případně znovu použít později. Špatné pořadí operací? Lze jej kdykoliv zaměnit.

„Na vstupu mám 100 000 záznamů, a právě tento záznam je pro mě poměrně důležitý. Jak se promítl do výsledku?“ Stačí kliknout na odpovídající číslo řádku, kde se záznam nachází a aplikace sama zobrazí jeho výslednou podobu včetně kontextu, ve kterém se nachází. To znamená, že pokud jsou použity nástroje, které zaměňují pořadí záznamu, pak ve výsledku při vyobrazení dvou a více záznamů budou vůči sobě v pozici, ve které se nachází ve výsledném textu. Tohle je další poměrně originální řešení, které nenabízí žádná z testovaných aplikací.

### Informovat uživatele

Uživatel by měl být vždy informován o tom, co právě provedl, co se v aplikaci právě děje nebo v jakém stavu se nachází. V aplikaci jsou proto implementovány různé funkce, které tuto situaci usnadňují.

Pro sdělující informace byla implementována vyskakující okna, která se zobrazují vlevo dole na monitoru a sdělují informaci po provedení určitých akcí. Celkem byly zavedeny čtyři typy, a to úspěšné (zelené), varovné (žluté), chybové (červené) a oznamující (modré) pro okamžité rozeznání kontextu. Zároveň je důležité, aby se okno nezobrazovalo příliš často a nestalo se tak pro uživatele obtěžující.

Další neméně důležitou vlastností je informovat uživatele o aktuálním stavu procesu zpracování textu. Jelikož práce s textovými řetězci je časově náročná operace, obzvlášť při velkém objemu dat, byla proto implementována načítající **deterministická** lišta, která vizuálně zobrazuje zbývající množství času, které je potřeba pro zpracování textu, aby uživatel nenabyl dojmu, že se nic neděje a aplikace zamrzla nebo neodpovídá. Hrubým příkladem při 10 000 záznamech nelze změnu postřehnout, ovšem při 200 000 ji už lze zaznamenat. Především záleží na počtu znaků v záznamu, náročnosti operace a jejich počtu.

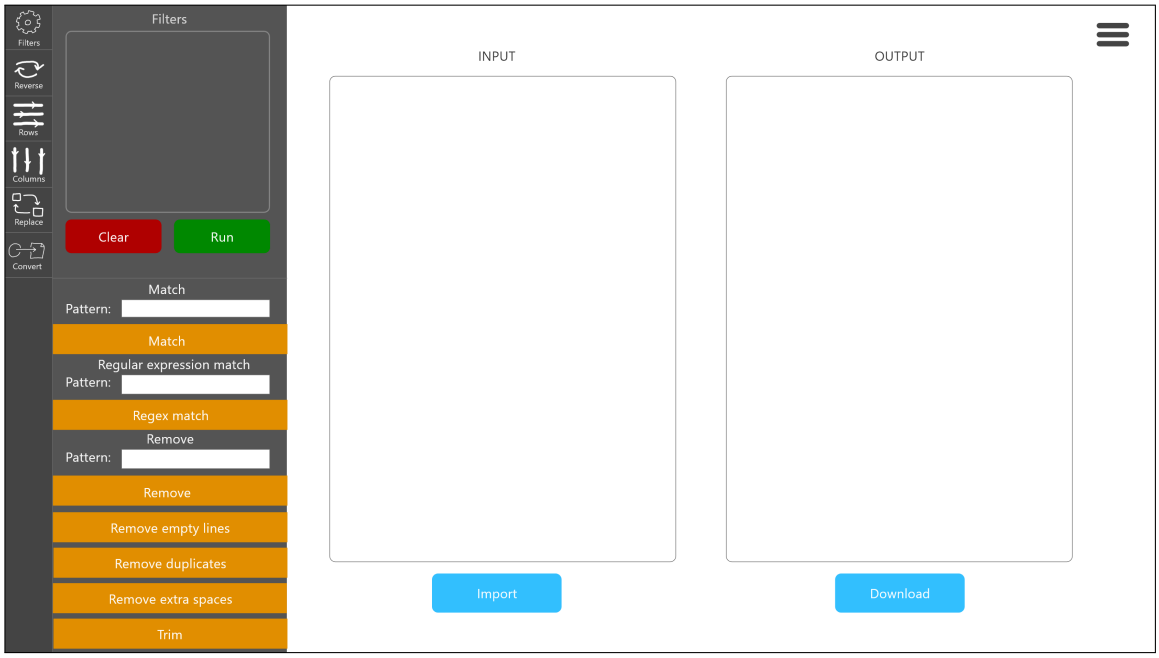
Všechny dostupné nástroje a možnosti jsou doprovázeny popisky, které vysvětlují význam operace nebo fungování nástroje, a tak by neměla nastat situace, kdy by uživatel nevěděl, co se bude dít. Tuto skutečnost potvrdilo i testování, které se nachází v kapitole 6.

O stavu editoru informuje spodní lišta, která zobrazuje:

- Pozici kurzoru v editoru
- Režim editoru
- Stav zalamování textu

## 3.2 Uživatelské rozhraní

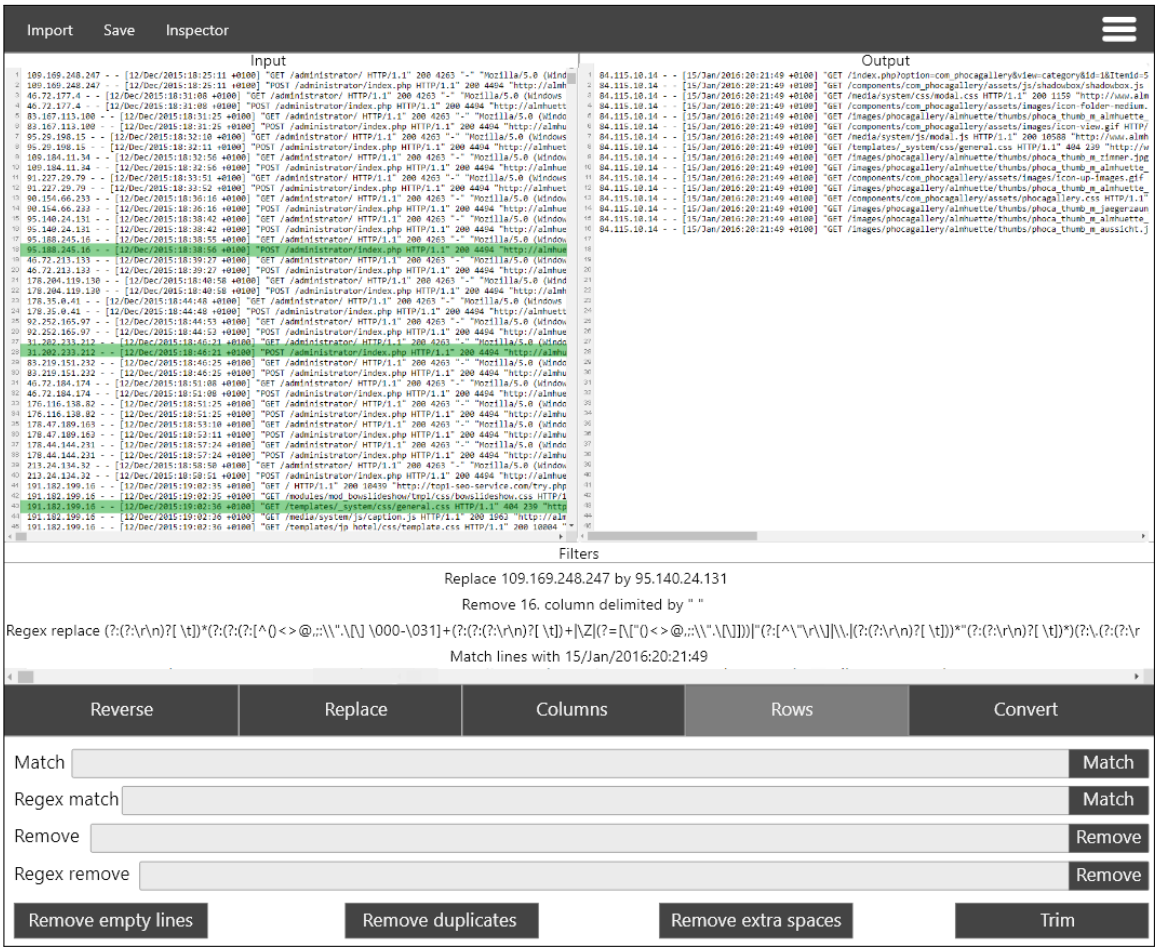
Na samotném začátku projektu byl vytvořen interaktivní prototyp aplikace (viz obrázek 3.1) v prostředí **AdobeXD**, na kterém byl demonstrován můj pohled na výsledný produkt.




Obrázek 3.1: První návrh aplikace.

Kategorie nástrojů se nachází vertikálně vlevo a po jejich volbě by se překreslily pouze nabízené nástroje, přitom horní okénko, které by zobrazovalo seznam použitých nástrojů by zůstalo zobrazené i nadále a obsahovalo by pouze názvy. Pod okénkem lze nalézt tlačítka pro spuštění převodu a odstranění filtrů.

Po předvedení návrhu a následné konzultaci se ukázalo, že návrh má silné nedostatky už v základním konceptu. Zejména editory omezují zobrazení textu, lepší by bylo eliminovat všechna prázdná místa okolo, editor maximálně rozšířit a umožnit uživateli si jeho velikost dynamicky přizpůsobit. Tím se dostáváme k problému s nástroji. Vstupy by měly být také širší, protože je vhodné u dlouhých textů mít možnost se snadno podívat na začátek a neřešit tento problém posuvníky. Špatné řešení poskytuje také seznam použitých filtrů, jelikož informace pouze o názvech je nedostačující, matoucí a při použití více nástrojů stejného typu není jasné, který provádí operaci v dané konfiguraci. Zároveň převod by se měl provádět automaticky a nenutit uživatele pokaždé dělat tento úkon manuálně. Tento návrh také neřeší možnost ladění, proto byl vytvořen nový návrh, který je zobrazen na obrázku 3.2.




Obrázek 3.2: Vylepšený návrh aplikace.

V novém návrhu byla galerie nástrojů přesunuta z levé části na spodní část. Tím bylo zajištěno více místa pro vstupy a popisky. Seznam použitých filtrů byl umístěn pod editor a také zabírá maximální horizontální prostor, tím je umožněno přehledně vypisovat všechny konfigurace. V tuhle chvíli byl navržen i styl ladění způsobem vyznačení řádků zelenou barvou, jak je znázorněno na obrázku 3.2. Přepínání mezi vyznačováním a klasickou editací by zajišťovalo tlačítko „inspektor“ v horní liště. Tento návrh poskytl solidní základ pro implementaci a další vylepšení bylo realizováno na základě testování a nově vzniklých situací při vývoji celé aplikace.

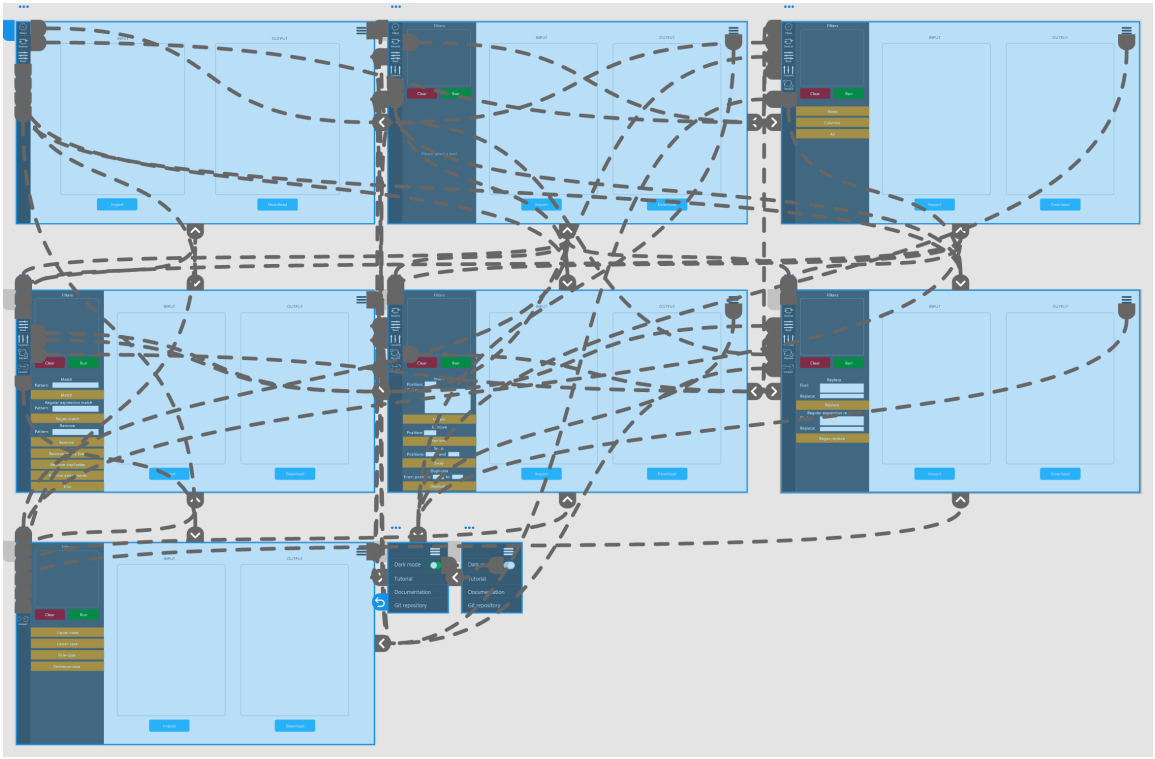
## AdobeXD

AdobeXD[1] [9] je první nástroj od společnosti Adobe navržený přímo pro vývojáře uživatelských rozhraní (dále jen UI). Využívá zkušeností UI návrhářů, kteří používali nástroje jako Photoshop nebo Illustrator a distribuuje je do jednoho produktu, jehož cílem je zvýšit jejich efektivitu při návrhu.

Designéři v tomto nástroji mohou snadno přepínat mezi návrhem a výsledným prototypem (viz obrázek 3.3). Veškeré změny v návrhovém režimu jsou tak hned dostupné v prototypovacím režimu, což usnadňuje definování aktivních bodů a přechodů, které oživují

[1] AdobeXD: https://www.adobe.com/cz/products/xd.html

maketu. Prototyp aplikace je možné snadno sdílet přes vygenerovaný odkaz a kdokoliv se na něj může podívat přes webový prohlížeč. Tím je umožněno snadné testování na různých typech zařízení jako jsou mobily, tablety nebo notebooky. Aplikace je zdarma k použití se základními fonty, omezením pouze na jeden sdílený prototyp a omezením cloudového úložiště na 2 GB.



Obrázek 3.3: Prototyp aplikace v AdobeXD.

## 3.3 Výběr nástrojů

Tato činnost byla součástí více etap vývoje. Rozhodování o tom, zda takový či jiný nástroj by měl být součástí aplikace není záležitostí časově řádu jednotek minut. Jedním z hlavních specifik a odlišností této aplikace je skutečnost, že nástroje lze postupně skládat, doplňovat navzájem, a tak by měly všechny funkce držet tento koncept. Proto mezi opakované typy otázek při výběru patří zejména:

- **Možnost řazení** – Použití nástroje na text v originální podobě by mělo umožnit i další zpracování.
- **Využitelnost** – Nástroj by neměl být příliš specifický vůči jakémukoliv oboru nebo profesi a měl by se držet prostého textu.
- **Skriptovací využití** – Nástroj by měl být vhodně interpretovatelný pro uživatele, kteří využijí vlastnost generování shell skriptů.
- **Zachování kontextu** – Konzistentní podoba před provedením operace musí zůstat i po jejím provedení. Jde o zachování možnosti uživateli poskytnout sledování původního textu.

## 3.4 Cílová skupina

Aplikace je koncipována do široké oblasti využití a snaží se problematiku spojenou filtrováním snížit na nejnižší možnou úroveň s cílem oslovit co největší množství potenciálních uživatelů. Zejména se může jednat o následující:

- **Studenti** – Často se setkávají s tabulkovými daty v projektech nebo při výpisech organizace.
- **Programátoři** – Úprava dat pro vstupy programů, analýza výstupu. Nejvíce asi ocení možnost generování kódu.
- **Administrativní profese** – Může se jednat o profese z ekonomické sféry nebo publikační agendy.
- **Vědečtí pracovníci** – Při zpracovávání vědeckých prací či ediční činnosti.
- **Ostatní uživatelé** – Nejsou příliš nuceni se setkávat s informacemi v této podobě a jde spíše o jednorázové využití.

# Kapitola 4

# Použité technologie

S rostoucí popularitou aplikací vzniklo velké množství balíčků (frameworků), které usnadňují programátorům práci při vývoji. Lze tak vytvářet nové aplikace podstatně rychleji, a proto nepoužít žádné dostupné řešení v projektu je dnes velmi neefektivní. V této kapitole budou představeny technologie použité při implementaci a popsán jejich účel, co nabízí a jak se s nimi pracuje. Také budou zmíněny alternativy dostupné na trhu a důvod, proč byla pro implementaci zvolena právě tato technologie.

## 4.1 Node.js

Node.js[1] [15] je open-source prostředí, které umožňuje psát aplikace na straně serveru v JavaScriptu a je postavené na JavaScriptovém enginu V8 od Googlu napsaném v C++. Přesto, že toto rozhraní bylo primárně vytvořeno pro serverové prostředí, používali ho vývojáři při vytváření nástrojů, které jim usnadňovalo práci při jejich úkonech. Od té doby se vyvinul zcela nový ekosystém nástrojů, který změnil způsob vývoje aplikací. Při implementaci byl z tohoto ekosystému použit nástroj pro správu balíčků, na který se následně zaměříme.

### Node Package Manager

Node Package Manager[2] (dále jen NPM) je open-source správce balíčků z prostředí Node.js. Nenabízí ovšem jen práci s balíčky, ale zahrnuje kompletní správu projektu od založení projektu až po jeho nasazení na web. Zejména se může jednat o překlad, sledování závislostí nainstalovaných balíčků a zranitelností aplikace, definování vlastních skriptů nebo ovládání lokálního serveru.

K vytvoření React webové aplikace (více o Reactu v následující kapitole 4.2) [4] je potřeba Node.js verze 8.10 a vyšší a NPM verze 5.6 a vyšší. Samotné vytvoření se provede příkazem `npx create-react-app myApp`, který provede vytvoření celého projektu a konfiguraci s tím spojenou. Znamená to, že veškeré nastavování je pro nás jako uživatele už vyřešené a lze se pustit do implementace. Ovšem v pozadí se toho děje mnohem více.

Jako první se začne tvořit soubor `package.json` [10], který obsahuje informace o použitých balíčcích (verze apod.), poté se provede samotná instalace. Jelikož aplikace zatím obsahuje pouze samotný React, nainstaluje pouze jeho knihovny jako React, React-dom a React-scripts. Po úspěšné instalaci se přesuneme k nainstalovaným skriptům, které jsou:

[1] Node.js: https://nodejs.org/en
[2] NPM: https://www.npmjs.com

- **start** – Spustí lokální server.
- **build** – Spustí překlad optimalizovaného kódu aplikace.
- **test** – Spustí testy.
- **eject** – Zpřístupní konfiguraci Reactu (jako je Babel, Webpack, ESLint atd.) v konfiguraci projektu. Vše co je provedeno v pozadí a typicky není potřeba dodatečně upravovat manuálně, ale mohou nastat situace, zejména ve větších projektech, kdy se tato vlastnost hodí.

Po úspěšné instalaci a konfiguraci se vytvoří struktura projektu, která je podrobněji popsána v kapitole 5.1.

### Alternativy

Mezi alternativy je vhodné zmínit open-source nástroj Yarn[3] vyvinutý společností Facebook. Oproti NPM je rychlejší a nabízí lepší řešení v oblasti zabezpečení, kdy v konfliktu dvou a více balíčků vznikají chyby v kódu. To byl také hlavní důvod vzniku tohoto nástroje.

## 4.2 React

Známé také jako ReactJS[4] [8] je open-source knihovna pro vytváření uživatelských rozhraní. Nejčastěji se používá při vývoji jednostránkových aplikací tzv. SPAs (Single Page Applications) a mobilních aplikací. Je optimalizována pro práci s rychle se měnícími daty a překreslováním. Nyní je spravována Facebookem a komunitou jednotlivých vývojářů a společností.

Velkou výhodou Reactu je možnost definovat části uživatelského rozhraní na menší celky (komponenty), ty hierarchicky skládat a znovu používat na více místech. Výsledek z určité úrovně granularity by pak mohl například vypadat jak na obrázku 4.1.

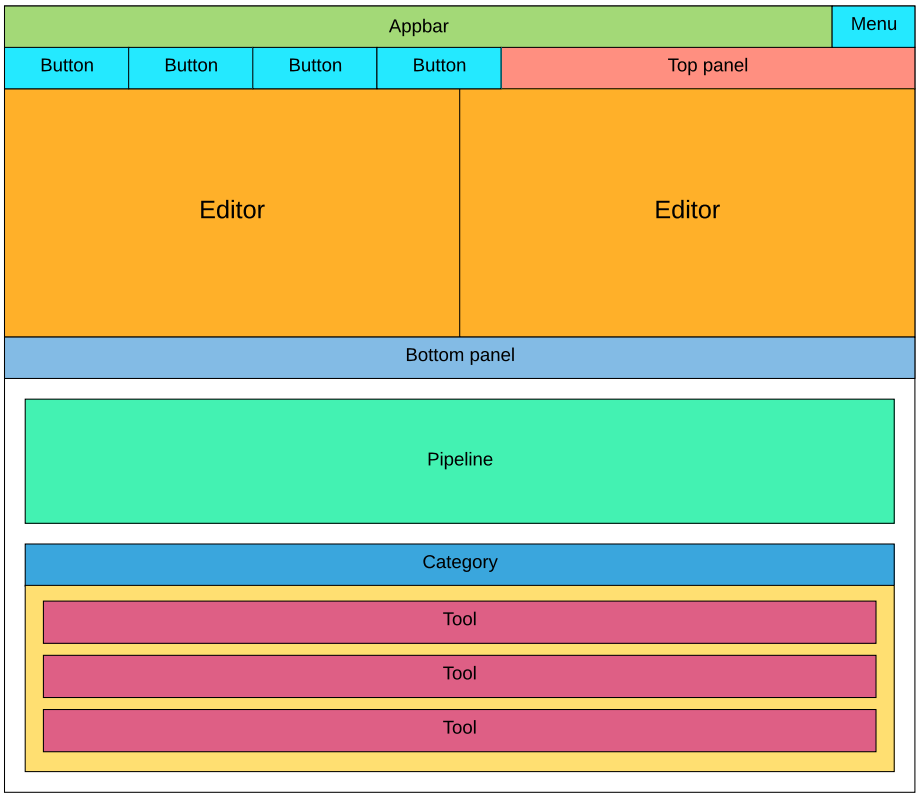
React je poměrně velká a komplexní knihovna, která nabízí nejen kvalitní řešení, ale poskytuje i větší pohodlí při vývoji. K zajištění takové funkcionality musí spolupracovat se spoustou dalších metod a balíčků. Mezi ty nejdůležitější patří:

- **JSX** – Syntax připomíná HTML, ale ve skutečnosti jde o rozšíření pro JavaScript k definování React prvků. Samotný React ho ovšem nevyžaduje, ale je velmi nápomocný.

```
const element = <h1>Hello, world!</h1>;
```

- **Babel**[5] – Protože React aplikace je napsána v JSX a webové prohlížeče této syntaxi nerozumí, vzniká problém. Tady přichází do hry Babel. Je to open-source JavaScriptový kompilátor, který převádí JSX zápis na prostý JavaScript.
- **Webpack**[6] – Je nástroj [1] pro zpracování souborů a usnadnění práce vývojářů. Jeho primárním účelem je vytvořit balíčky z modulárního JavaScriptového kódu pro jeho

[3] Yarn: https://yarnpkg.com
[4] React: https://reactjs.org
[5] Babel: https://babeljs.io
[6] Webpack: https://webpack.js.org




Obrázek 4.1: Dekompozice rozhraní aplikace na menší komponenty.

použití v prohlížeči. Neomezuje se ovšem pouze na JavaScriptové soubory, ale umožňuje zpracovat téměř jakýkoliv druh jako jsou obrázky, serializační formáty aj. Výstupem je modulární JavaScriptový balíček.

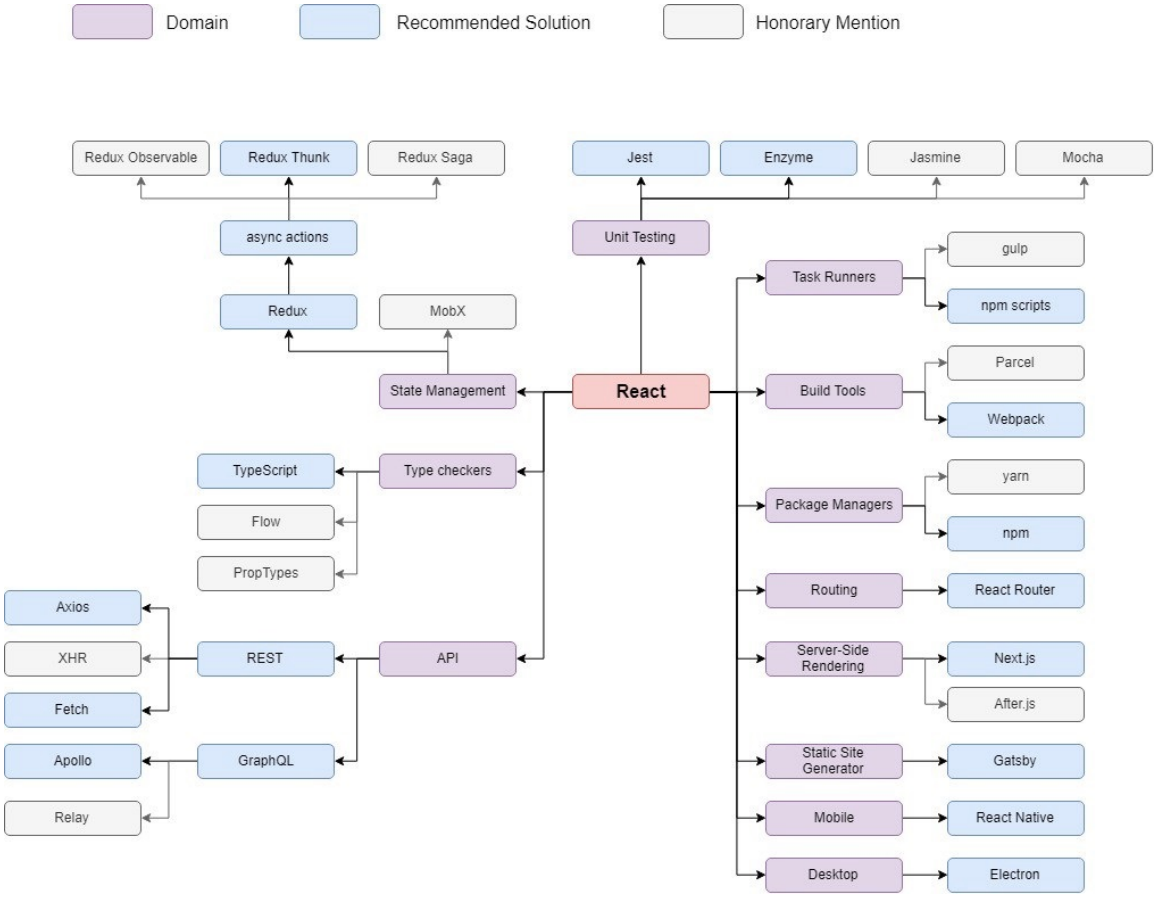
Další balíčky, se kterými může React pracovat, jsou znázorněny na obrázku 4.2. Ve skutečnosti se všemi najednou nepracuje, záleží především na požadavku výsledné aplikace.

### Vznik a historie

ReactJS [2] začal jako JavaScriptový port XPH od Facebooku při řešení problému s minimalizací útoku cross-site scripting (XSS). Tento typ útoku spočívá ve využití chyb ve skriptech běžících na webu (především neošetřené vstupy). Útočník může takto do stránek podstrčit vlastní JavaScriptový kód, který dokáže poškodit vzhled stránky, znefunkčnit ji nebo získat citlivé údaje návštěvníků. Tato metoda je též často využívána při phishingu. Nastal však problém, kdy dynamické webové stránky poskytují dlouhou zpáteční cestu na server a XHP tento problém nevyřeší. Došlo tak k zavedení XHP do prohlížeče pomocí JavaScripu. Výsledkem byl React.

### Objektový model dokumentu

Objektový model dokumentu [13] (dále jen DOM) z anglického názvu Document Object Model je multiplatformní a jazykově nezávislé rozhraní reprezentující HTML, XHTML nebo XML jako logickou stromovou strukturu.



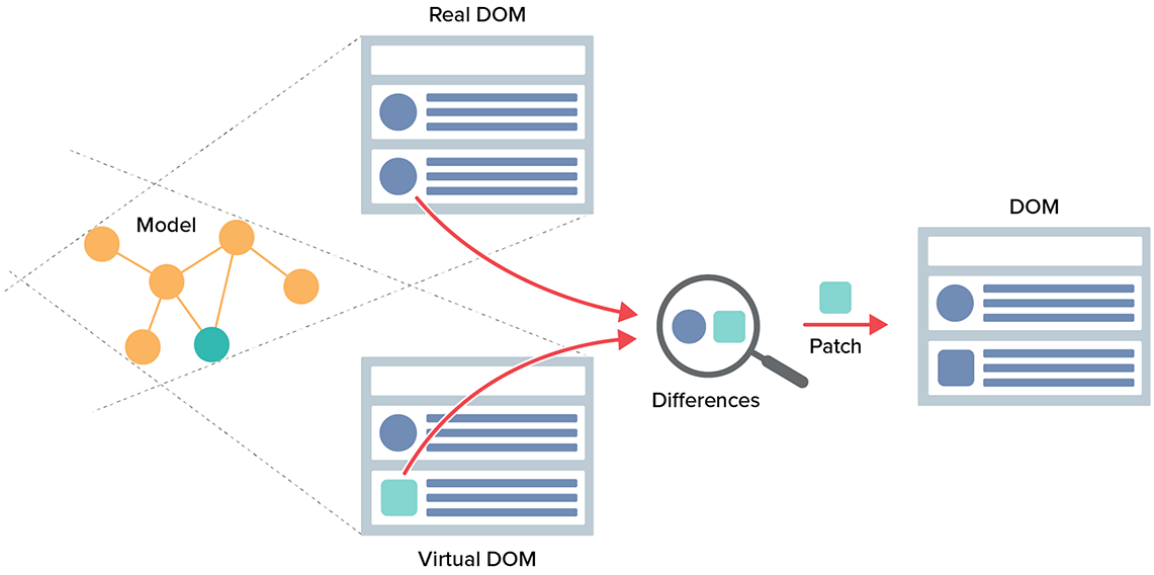

Obrázek 4.2: Podmnožina ekosystému Reactu. Zde jsou znázorněny některé známé, často používané balíčky. Výsledný ekosystém je znatelně větší. Převzato z [8].

React [1, 13] vytváří v mezipaměti strukturu zvanou **Virtuální DOM**, ve které vypočítává provedené změny a poté aktualizuje DOM prohlížeče. Tato vlastnost umožňuje programátorům naprogramovat celou aplikaci se závislostmi, při kterých je potřeba překreslení. Knihovna poté bude tyto části aplikace překreslovat pouze, pokud je doopravdy provedena změna, a je tak umožněno aktualizovat pouze jeden DOM element, který se změnil. React provádí proces aktualizace ve třech krocích:

1. Po změně prvku se překreslí celý virtuální DOM Reactu.
2. Provede se porovnání s předchozí DOM reprezentací.
3. Skutečný DOM prohlížeče bude aktualizován na základě změn porovnání. Tento krok může připomínat použití opravného patche (viz obrázek 4.3).

### Komponenty

Komponenty [13] jsou modulární části kódu, které umožňují rozdělit uživatelské rozhraní na nezávislé, opakovaně použitelné kusy. Koncepčně jsou komponenty jako JavaScriptové funkce, které přijímají libovolné vstupy (props) a vracejí prvky Reactu, které popisují výslednou podobu na obrazovce. Mezi dobré zásady patří skládání komponent do logické



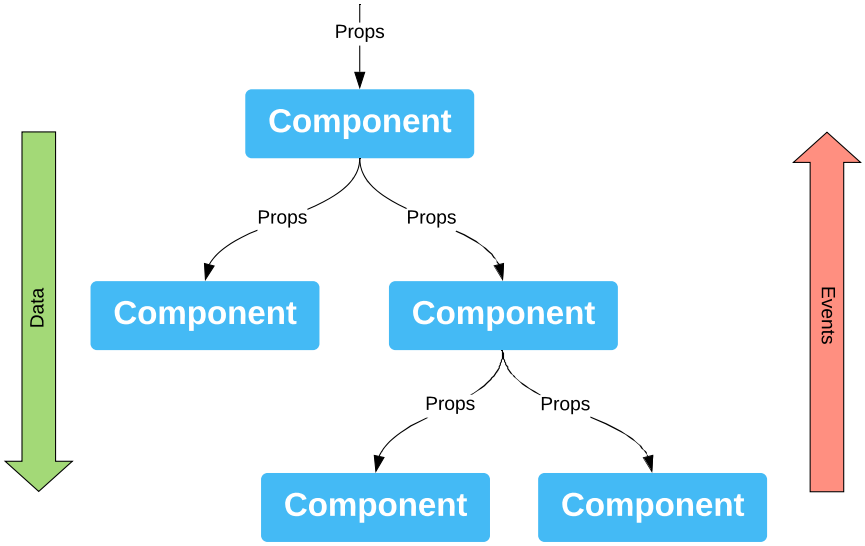

Obrázek 4.3: Proces aktualizace struktury DOM v Reactu. Převzato z [13].

stromové struktury předávající si parametry z nadřazené komponenty k podřízené, tedy směrem shora dolů tak, jak je znázorněno na obrázku 4.4. Pokud nadřazená komponenta bere vstup z podřízené přes referenci, typicky tato situace indikuje, že je dostupné lepší řešení. Ovšem existují situace, kdy takové řešení je nejlepší, například u přehrávání animací nebo při použití externích knihoven.

Chování aplikace neboli komponenty v daném okamžiku je definováno jejím aktuálním stavem. Stav obsahuje data, která se mění na základě interakce s uživatelem, obvykle obsluhou událostí. Jinak řečeno, když se změní stav komponenty, React znovu vykreslí odpovídající část do prohlížeče. Vstupy komponenty jsou na rozdíl od stavu neměnné a nastavují se nadřazenou komponentou. Komponenty můžeme rozdělit podle stavu na:

- **Bezstavové**
  - Nemají žádný stav
  - Mají vstupy
  - Překreslení je závislé na vstupech
- **Stavové**
  - Mají měnící se stav
  - Stav nelze číst a nastavit mimo komponentu
  - Mají vstupy
  - Překreslení je závislé na vstupech a stavu

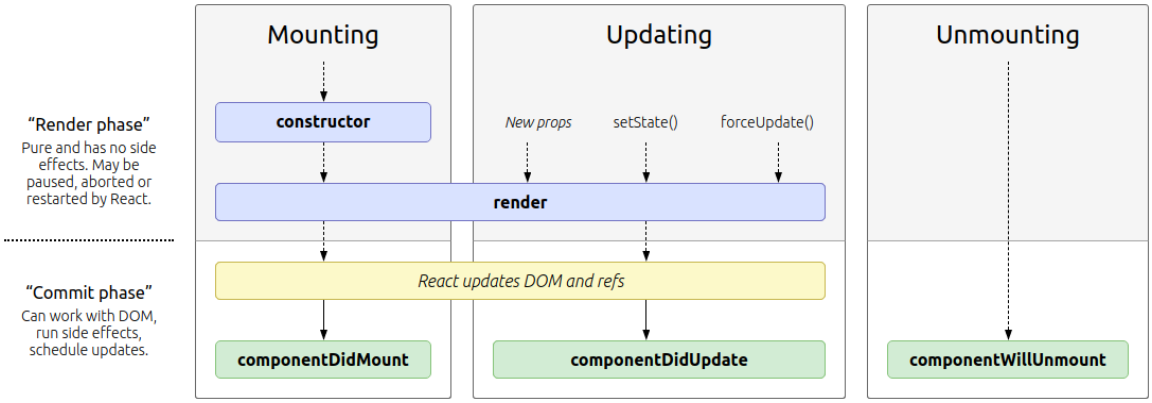
Samotné komponenty lze implementovat ve dvou zápisech, a to třídním nebo funkcionálním a je vhodné se tohoto zápisu držet v celém projektu. Nejviditelnějším rozdílem je syntax a také, že Babel v případě třídní komponenty transponuje více kódu. Funkcionální komponenty jsou snadněji čitelné než třídní, protože připomínají spíše funkce. Od verze 16.8 disponují funkcionální komponenty také hooky a jsou tak funkčně ekvivalentní s těmi třídními. Dříve se používaly pouze u bezstavových komponent.




Obrázek 4.4: Hierarchie React komponent ve stromové struktuře.

### Životní cyklus komponent a React hooks

Životní cyklus komponent [13] je znázorněn na obrázku 4.5 a probíhá ve třech fázích: montování, aktualizace a odpojování. V těchto fázích probíhají metody zvané „hooks“, ke kterým dochází v určitém pořadí jak je znázorněno na obrázku 4.6.




Obrázek 4.5: Fáze životního cyklu komponenty. Převzato z [13].

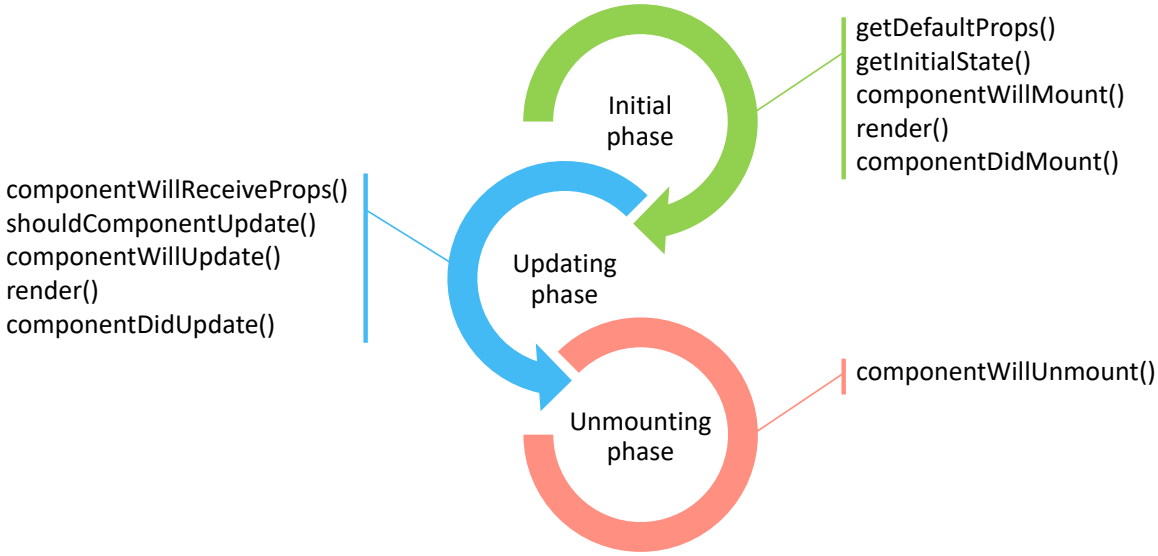
- **Montování**
  1. `constructor()` – Zde lze inicializovat počáteční stav komponenty a navázat metody pro obsluhu událostí.
  2. `componentWillMount()` – Metoda je zavolána před započetím montování.
  3. `render()` – První vykreslení, jedná se tedy o montování komponenty.
  4. `componentDidMount()` – Metoda je zavolána po ukončení montování.

- **Aktualizace**

  1. `componentWillReceiveProps()` – Metoda je zavolána u namontované komponenty před obdržením nových vstupu.
  2. `shouldComponentUpdate()` – Ve výchozím režimu React překreslí komponentu kdykoliv se změní stav nebo vstup. Tato metoda oznamuje, že výstup komponenty není závislý na daném typu dat, a tak není nutné překreslení. Slouží jako optimalizační metoda.
  3. `componentWillUpdate()` – Metoda je zavolána vždy před započetím překreslení.
  4. `render()` – Překreslení komponenty.
  5. `componentDidUpdate()` – Metoda je zavolána okamžitě po překreslení komponenty. Neplatí pro první vykreslení.

- **Odpojování**

  1. `componentWillUnmount()` – Metoda je zavolána před odpojením a zničením komponenty. Běžně se používá pro uvolnění alokovaných zdrojů.




Obrázek 4.6: Výskyt hooků ve fázích životního cyklu komponenty. Inspirováno z [13].

V případě funkcionálních komponent se využívají jiné hooky, které nabízí podobnou funkcionalitu jako předchozí zmíněné. Jelikož aplikace je implementována ve funkcionálním zápisu, tak zde jsou uvedeny alespoň ty nejpodstatnější:

- `useState()` – Slouží k definici stavu komponenty. Jelikož se jedná o funkce, je nutno explicitně říct Reactu, na základě kterých stavů má být provedeno překreslení.

- `useEffect()` – Tento hook je zavolán vždy po změně stavu komponenty s možností definovat stavy, které mají být sledovány. Pokud není definováno, chová se jako metoda `componentDidUpdate()`. Nabízí také tzv. „clean-up effect“ neboli vyčištění, ve kterém lze po provedení metody uvolnit alokované zdroje. V takové kombinaci lze dosáhnout i chování `componentDidMoount()` nebo `componentWillUnmount()`.

- `useCallback()` – Optimalizační metoda. Slouží pro definování stavů, které ovlivní přetvoření funkce. Ve výchozím režimu, vždy po překreslení komponenty se přetvoří i funkce, které obsahuje, i když se jejich kód nezměnil, to pak způsobí překreslení všech následujících komponent, které přijímají tuhle funkci jako vstup.
- `useRef()` – Umožňuje odkazovat podřízenou komponentu a přistoupit tak k jejím instancím.
- `useImperativeHandle()` – Definuje instance, které budou zpřístupněny nadřazené komponentě. Takto nadřazená komponenta pak může k těmto instancím přistoupit přes referenci.
- `useMemo()` – Slouží k memorování předchozího průběhu. Typicky se tato metoda využívá na místech, kde je kód často prováděný (např. funkce) k urychlení výpočtu. Není vhodné tuto metodu používat všude, ale jen na určitých místech, kde dochází k viditelnému zpomalení.

### Shrnutí a alternativy

React je v současnosti nejpopulárnější knihovna svého druhu, která nabízí spoustu možností a stále se rozrůstá. Používají ji velké společnosti jako Facebook, Instagram nebo Netflix. Nabízí velkou rychlost zpracování díky implementaci virtuální struktury DOM. Další podstatnou výhodou je, že volba konceptu programování je ponechána na uživateli mezi funkcionálním a třídně založeným stylem. Oproti tomu jsou zde i nevýhody jako je dělení kaskádových stylů do zvláštních souborů nebo zavedení přímo do komponent a míchání logiky programu se šablonami, což může být pro některé uživatele matoucí.

Alternativou může být knihovna **Angular**[7] od společnosti Google, kterou využívají například společnosti Microsoft, Apple a AT&T. Angular [14] přebírá mnoho konceptů z návrhového vzoru Model-View-ViewModel (MVVM), který umožňuje přehledně oddělit logiku od rozhraní aplikace a má mimořádnou podporu TypeScriptu. Struktura a architektura je navržena speciálně pro velkou škálovatelnost projektu. Mezi nevýhody patří poměrně malý výkon a poměrně vysoká složitost.

Další v dnešní době velmi známou alternativou je knihovna **VueJS**[8], za kterou nestojí žádná velká společnost a využívají ji společnosti jako GitLab, Adobe a Xiaomi. Knihovna [14] nabízí detailní dokumentaci a umožňuje vytvářet znovupoužitelné šablony. V mnoha ohledech si bere inspiraci z Reactu a Angularu, to umožňuje snadnější přestup mezi knihovnami. Mezi nevýhody patří nedostatečná podpora integrace do větších projektů a menší ekosystém oproti dříve zmíněným knihovnám.

## 4.3 Material-UI

Material-UI[9] je dnes jeden z oblíbených React frameworků pro tvorbu uživatelských rozhraní, který vznikl v roce 2014 na implementaci material designu[10]. Mezi společnosti využívající Material-UI patří např. Bethesda, Amazon a Unity. Framework nabízí předdefinované

[7] Angular: https://angular.io
[8] VueJS: https://vuejs.org
[9] Material-UI: https://material-ui.com
[10] Material design: https://material.io

části rozhraní (elementy) jako React komponenty s cílem usnadnit a zrychlit tvorbu a zároveň zajistit vyšší míru kvality. V současné verzi 4.9.8 má Material-UI přes 5 milionů stažení měsíčně přes správce balíčku NPM (viz kapitola 4.1).

Material-UI poskytuje také balíček s ikonami, který lze dodatečně přidat. Sada nabízí přes 1 100 ikon, které jsou ve většině případů zaměřené spíše na mobilní zařízení, přesto v tomhle projektu je sada postačující, a tak není nutné importovat další. Způsob zacházení je potom stejný jako s ostatními prvky Material-UI nebo komponenty.

### Material design

Jde v zásadě o filozofii designu (tzv. vizuální jazyk), kterou Google [11] poprvé představil v roce 2014 na konferenci Google I/O. Primárně je určen pro mobilní platformu, ale principy se přenesly napříč všemi zařízeními. Material design poskytuje návod na podobu rozhraní od rozvržení přes animace až po barvy. Samotný koncept je inspirován reálným světem a jeho texturami, včetně toho, jak se odráží světlo a vznikají stíny na povrchu materiálů, odtud byl přejat název „material“.

### Alternativy

Známou alternativou a jednou z nejstarších je knihovna Bootstrap, přesněji pro React knihovna React-Bootstrap[11], která poskytuje prvky jako React komponenty bez dodatečných závislostí (třeba na jQuery). Výhodou je soulad mezi prohlížeči, takže by se stránka měla vždy korektně vykreslovat a zároveň není náročná. Mezi nevýhody patří špatná přizpůsobitelnost, což může vést k podobám mezi weby, které spolu ani nemusí souviset.

## 4.4 Ace editor

Jak je zmíněno v kapitole 3.1, editor je základ celé aplikace, a proto by měl nabízet také mnoho funkcí. S takovými je možné se setkat například v programování. Editor, který poskytne programátorovi největší komfort se stane velmi oblíbený. Tady se naskytuje možnost tento editor využít a předělat ho na potřeby této aplikace.

Jedním z těchto editorů je open-source Ace[12] poskytovaný s BSD licencí, jedná se tedy o svobodný software s volným šířením. Ace je napsán v JavaScriptu a udržován jako primární editor pro Cloud9 IDE[13] a je nástupcem projektu Mozilla Skywriter[14] (dříve Bespin). Nyní je Ace jedním z nejpopulárnějších webových editorů vůbec a je používán v projektech jako Codebender, Overleaf nebo Wikipedia.

Samotný Ace nabízí vysoký výkon pro editaci textu, oficiální stránka udává limit na 4 milióny řádků, ale záleží na průměrném počtu znaků na řádek. Dále je možné využít zvýrazňování syntaxe, přes 20 motivů, číslování řádků, klávesové zkratky a další. Nejzajímavější vlastností je ovšem ladící rozhraní určené pro programátory, které bylo přizpůsobeno na potřeby této aplikace. Při implementaci byl použit oficiální wrapper[15] pro React, který zpřístupňuje Ace jako komponentu. Další práce je tak umožněna přes vstupy nebo referenci. V opačném případě by bylo potřeba vytvořit vlastní wrapper.

[11] React-Bootstrap: https://react-bootstrap.github.io
[12] Ace editor: https://ace.c9.io
[13] Cloud9 IDE: https://aws.amazon.com/cloud9
[14] Mozilla Skywriter: https://www.mozillalabs.com/en-US/skywriter/index.html
[15] Ace wrapper: https://github.com/securingsincity/react-ace

### Alternativy

Druhý v dnešním trendu je editor **CodeMirror**[16], který v aktuální verzi 5.52.2 nabízí přehlednou dokumentaci a velmi podobnou funkcionalitu jako výše zmíněný Ace. Je používaný v projektech jako Adobe Brackets, JSFiddle nebo GitHub. Další vhodnou alternativou je **Monaco**[17] vyvinutý společností Microsoft. Editor nabízí moderní vzhled a mini-mapu celého dokumentu, která usnadňuje orientaci. Mezi nevýhody patří nepodporované mobilní zařízení a poměrně malá sada poskytnutých nástrojů oproti výše zmíněným editorům. Projekty, které využívají Monaco jsou např. Visual Studio Code nebo CodeSandbox.

## 4.5 Web worker

Web worker API [12] je JavaScriptový kód běžící v pozadí aplikace, bez narušení výkonu stránky. Obecně JavaScript je jednovláknový a může provádět jen jednu operaci v daný okamžik. Zde však vzniká problém s výpočetně náročnými operacemi nebo operacemi nad obsáhlými daty, protože i během tohoto výpočtu je potřeba neustále reagovat na události od uživatele, jinak by mohla nastat situace, kdy se uživatel snaží provést úkon ve své úloze, ale aplikace nereaguje, protože počítá výsledek. V tomto případě je zapotřebí zavést pojem vícevláknové zpracování.

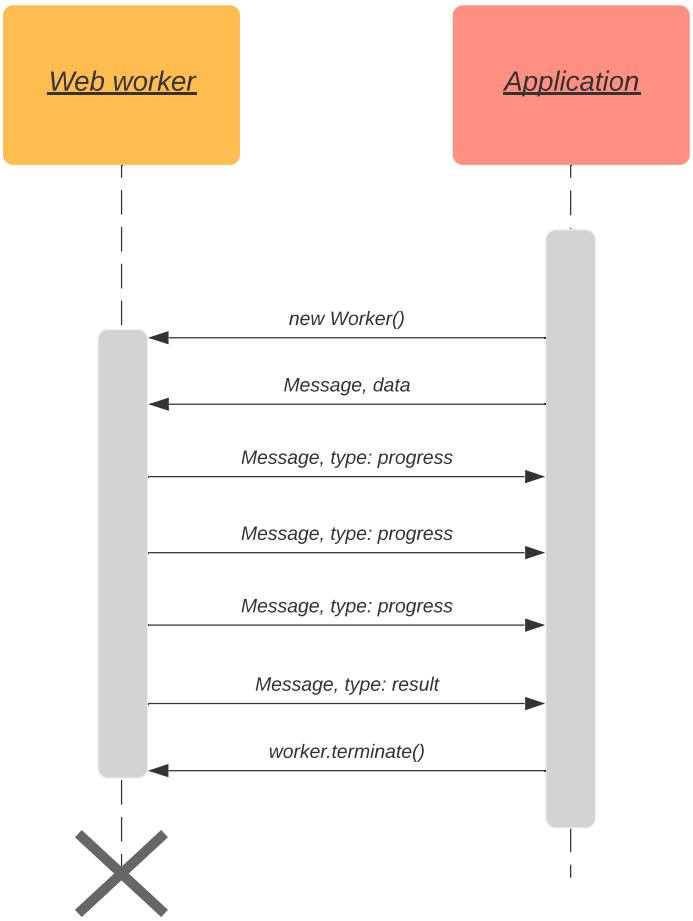
API komunikuje s hlavní aplikací pomocí metody zasílání zpráv. Hlavní aplikace pošle požadavek skriptu a následně dostává odpovědi. Během procesu zpracování se pak může dále věnovat požadavkům uživatele. V případě této aplikace je řešení této problematiky nutné z důvodu, že data získána od uživatele mohou být rozsáhlá a také zpracování textových operací je poměrně náročné, jak bylo již dříve zmíněno v kapitole 3.1.

Konkrétně vždy, když je potřeba data zpracovat, odešle hlavní aplikace zprávu s daty a informacemi o způsobu jejich zpracování, ty obsahují například použité nástroje a jejich konfiguraci, stav editoru aj. Vlákno poté začne data zpracovávat, informuje o aktuálním stavu procesu převodu hlavní aplikaci a nakonec ji pošle výsledek. Tím se dostáváme k zavedení typu zpráv. Během získávání odpovědí do aplikace je potřeba rozlišit, jak interpretovat danou odpověď. Pokud se jedná o odpověď nesoucí informaci, že proces transformace je například z 80% kompletní, je tato informace sdělena uživateli, pokud se jedná o odpověď obsahující výsledek převodu, tak je tento výsledek zobrazen. Proto každá zpráva musí obsahovat typ, který identifikuje její obsah. Proces komunikace pak může vypadat jak na obrázku 4.7.

## 4.6 Selenium

Selenium[18] [3] je open-source platformově nezávislý nástroj implementovaný v Javě pro automatické testování webových aplikací nebo e-shopů v různých prohlížečích. Samotné testy je možné vytvářet interaktivně pomocí Selenium IDE[19] v prohlížeči nebo pomocí programovacích jazyků jako jsou Java, Python, Perl, PHP nebo Ruby. Selenium může být zároveň integrován nástroji pro správu testů.

[16] CodeMirror: https://codemirror.net
[17] Monaco: https://microsoft.github.io/monaco-editor
[18] Selenium: https://www.selenium.dev
[19] Selenium IDE: https://www.selenium.dev/selenium-ide




Obrázek 4.7: Komunikace mezi aplikací a web workerem.

### Sada nástrojů

Selenium také poskytuje následující sadu nástrojů, které usnadňují testování:

- **Selenium IDE** – Je dostupné jako rozšíření do webového prohlížeče Firefox nebo Chrome. Toto prostředí nabízí nejjednodušší způsob pro tvorbu testů jednoduchou uživatelskou interakcí v prohlížeči a následným vygenerováním skriptu provádějící tuto sekvenci událostí. Tato metoda nevyžaduje znalost programování.
- **Selenium RC** – Jedná se o proxy server pro instance webových prohlížečů, který přijímá příkazy z podporovaných jazyků uživatelských programů a předává je do prohlížeče. Po zprovoznění serveru se k němu klienti připojují a spouští své testy, jedná se o klasické klient-server řešení.
- **Selenium WebDriver** – Umožňuje pracovat lokálně přímo s aplikacemi prohlížečů. Není proto nutné nastavovat Selenium server, nicméně je nutné nastavit ovladač pro každý prohlížeč zvlášť.
- **Selenium Grid** – Umožňuje [3] spouštět testy paralelně na několika počítačích v různých prostředích na různých operačních systémech. Jedná se o rozbočovač připojený k několika uzlům. Přijatý test je poté předán uzlu, který odpovídá požadavkům (platforma a prohlížeč), ve kterém má být spuštěn.

## 4.7 GitHub Pages

GitHub Pages[20] je webhostingová služba poskytovaná od roku 2008 společností GitHub, u které byl projekt také verzován během implementace a následně zveřejněn. Výchozí stránky jsou dostupné pod doménou `github.io`, ale je možné nastavit i vlastní. V roce 2016 začaly být stránky poskytovány pod šifrovaným protokolem HTTPS. Tato služba [7] nesmí být ovšem využívána jako bezplatný webhosting k provozování online obchodu, komerčních webů nebo jiných webů, které jsou primárně určeny k obchodním transakcím nebo poskytování komerčního softwaru (SaaS). Mezi další limity používání pro snížení zátěže na servery patří:

- Publikovaný obsah na webu nesmí být větší než 1 GB.
- Omezená šířka pásma na 100 GB za měsíc.
- 10 sestavení (builds) za hodinu.

Při překročení těchto kvót je uživatel informován e-mailem od podpory s návrhy pro budoucí provoz jako může být například přechod na jinou platformu.

[20] GitHub Pages: https://pages.github.com

# Kapitola 5

# Implementace webové aplikace

Snaha o to, aby aplikace byla využívána a snadno dostupná bylo potřeba zvolit správnou platformu, proto už od začátku bylo zřejmé, že aplikace bude implementována ve webovém rozhraní. Jedná se o aplikaci, která běží pouze na straně klienta (tzv. „client-side“), proto byl pro implementaci zvolen programovací jazyk JavaScript.

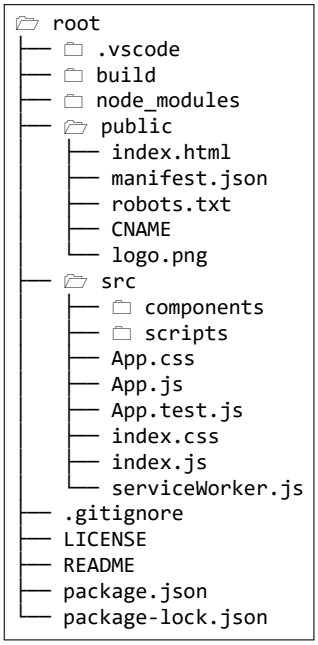
V táto kapitole budou popsány podstatné informace k pochopení logiky implementace, detailněji budou zmíněny klíčové části kódu a taktéž ukázány pokročilejší techniky při řešení problémů, které implementace obnášela. Dále je popsán postup zveřejnění aplikace na web a potřebná konfigurace při tomto procesu. Konec této kapitoly je věnován výsledné podobě implementovaného uživatelského rozhraní.

## 5.1 Struktura projektu

Pro případnou budoucí práci nebo k hlubšímu pochopení jak React nebo web funguje je vhodné zmínit strukturu projektu a popsat, jak tyto části pracují a jaký je jejich účel. Z důvodu poměrně velkého rozsahu projektu jsou zanedbány méně podstatné části, jejichž podstata není příliš důležitá nebo lze jejich účel analogicky odvodit (třeba z již dříve zmíněných informací). Na obrázku 5.1 je znázorněna tato adresářová struktura projektu a funkční popis je následující:

- **.vscode** – Obsahuje konfiguraci editoru Microsoft Visual Studio Code, ve kterém byl projekt napsán. Je zde definováno např. nastavení ladícího režimu vůči lokálnímu serveru, který běží na portu 3000.
- **build** – Je adresář, do kterého bude umístěn finální zkompilovaný a optimalizovaný kód provedením příkazu `npm run build`. Obsah je pak připraven na nasazení na web bez nutnosti další interakce.
- **node_modules** – Je místo pro nainstalované balíčky pomocí správce balíčků (např. NPM). Adresář není potřeba přenášet (např. na GitHub), protože kdokoliv, kdo si stáhne zdrojové kódy projektu, si může tyto balíčky stáhnout přímo (v závislosti na souboru `package.json` nebo `package-lock.json` viz níže)
- **public** – Tento adresář obsahuje místo, ke kterému je možné přistoupit „zvenčí“. Soubory uvnitř nebudou nijak modifikovány kompilátorem a ve výsledku budou v kořenovém adresáři sestavené aplikace. Typicky mezi takové soubory patří výchozí soubor `index.html`.

- **index.html** – První soubor navštívený webovým prohlížečem. Soubor v těle obsahuje pouze jeden element s identifikátorem `root`, na který je namapována celá aplikace.
- **manifest.json** – Obsahuje popis aplikace, který je použit na mobilních zařízeních, pokud je aplikace přidána na domovskou obrazovku.
- **robots.txt** – Soubor obsahuje pravidla přístupu nebo blokování k souborům pro jednotlivé uživatelské agenty (roboty) např. Googlebot.
- **CNAME** – „Canonical name“ obsahuje alias, který byl vytvořen při nastavování vlastní domény (více v kapitole 5.5 o nasazení aplikace).




Obrázek 5.1: Adresářová struktura projektu.

- **src** – V tomto adresáři je umístěna z velké části samotná implementace aplikace jako jsou komponenty nebo skripty. Struktura není pevně daná, proto hierarchii podsložek lze libovolně upravovat pro lepší orientaci.
  - **components** – Zde jsou umístěny komponenty aplikace, které tvoří logický strom (více o dekompozici komponent v následující kapitole 5.2).
  - **scripts** – Obsahuje skripty nutné pro běh aplikace, patří sem např. generátor shellu, logika transformace, web worker atd.
  - **App.js** – Komponenta na nejvyšší úrovní, ze které se větví další komponenty umístěné v adresáři `components`.
  - **index.js** – Slouží jako „brána“ mezi stromem komponent vzniklý z kořene `App.js` a `root` elementem v souboru `index.html`.
  - **serviceWorker.js** – Je skript [5] běžící na pozadí, oddělený od webové stránky a otvírá dveře funkcím, které nevyžadují interakci s webovou stránkou nebo uživatelem. Dnes obsahuje funkce jako jsou rozesílání push notifikací, chytrá práce s mezipamětí prohlížeče, synchronizace na pozadí aj. V aplikaci je tento

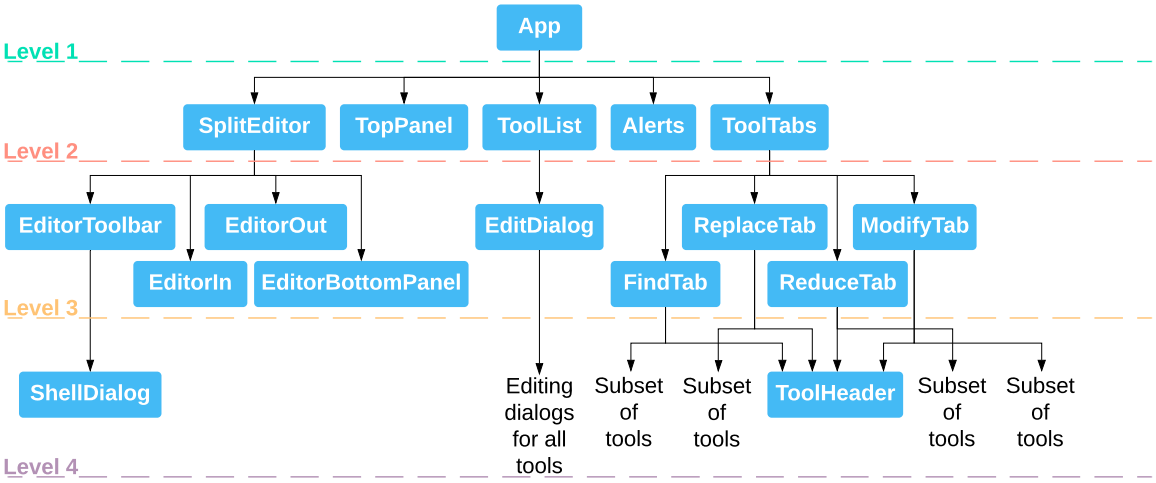
skript vygenerován, nezměněn a nezaregistrován, jde spíše o doplnění pro budoucí použití.

- **package.json** – Slouží nejen pro ukládání přímých závislostí na balíčcích, ale také usnadňuje jejich správu a instalaci. Nachází se zde i další metadata projektu jako je jméno aplikace, verze, licence apod. Zajímavou sekcí je sekce se skripty, která umožňuje definovat vlastní příkazy pro NPM, které byly využity například při nasazení aplikace (viz kapitola o nasazení 5.5).

- **package-lock.json** – Je automaticky generovaný soubor NPM při modifikaci `node_modules` nebo `package.json`, který obsahuje přesný strom závislostí verzí, to zaručuje shodné závislosti pro další programátory a definuje chování programu. Ve výchozím nastavení `package.json` obsahuje pouze přímé závislosti, nikoliv závislosti závislostí, proto i když zamkneme strom přímých závislostí není zaručeno, že plný strom bude identický. Vhodné je ponechat tento soubor ve správě NPM a manuálně do něj nezasahovat.

## 5.2 Dekompozice aplikace

Dekompozice aplikace do funkcionálních komponent Reactu je znázorněna ve stromové struktuře na obrázku 5.2, kde jména jsou odvozena od názvů souborů v projektu. Každá komponenta je zodpovědná za určitou část aplikace a případně drží potřebná data. V téhle situaci může být na místě otázka, zda nepoužít některou z knihoven pro řízení stavu, které jsou součásti ekosystému Reactu, jako je například knihovna Redux. Když se podíváme znovu na obrázek 5.2 vidíme, že výška stromu je pouze 4 a zároveň data potřebná přenášet napříč celou aplikací je relativně málo, proto zavedení tohoto dodatečného frameworku mi přišlo poměrně těžkopádné. V případě budoucí větší expanze aplikace (z pohledu dat nebo výšky stromu) je možné knihovnu pro řízení stavu snadno přidat a přesunout do ní tato data. Stručný popis ke komponentám je potom v následujícím textu.




Obrázek 5.2: Soustava komponent aplikace ve stromové struktuře. Každá komponenta obsahuje část rozhraní aplikace, případně drží potřebná data. Množina podobných komponent na listové úrovni je nahrazena textem kvůli jejich velkému počtu.

- **App** – Jak již bylo zmíněno v předchozí kapitole 5.1 jedná se o komponentu na nejvyšší úrovní a jsou zde uložena zejména data potřebná napříč celou aplikací jako je např. seznam použitých nástrojů a jejich konfigurace, protože pokud dvě a více komponent potřebují určitá společná data, je potřeba tyto data definovat v nejbližší, společné, nadřazené komponentě. Dále se zde nachází inicializace web workera a správa potřebného kontextu pro provedení transformace (nastavení editoru, breakpointů aj.).
- **TopPanel** – Obsahuje horní panel s názvem aplikace a menu.
- **SplitEditor** – Slouží jako obálka pro editory. Nachází se zde nastavení zalamování řádků, přizpůsobení velikosti, ukazatel průběhu a další.
- **EditorToolbar** – Obsahuje panel nástrojů pro editor. Jsou zde stavy upřesňující nastavení hledaní pomocí regulárních výrazů nebo citlivost na velká písmena, jinak komponenta spíše využívá metody nadřazených komponent.
- **ShellDialog** – Zde je definováno okno pro výpis shell skriptu pro provedení transformace na Linuxu. Samotné generování skriptu je pak provedeno v souboru `shellGenerator.js`, který je odtud zavolán.
- **EditorIn** – Zahrnuje vstupní editor a konfiguraci breakpointů.
- **EditorOut** – Obsahuje výstupní editor.
- **EditorBottomPanel** – Komponenta pouze zobrazuje aktuální konfiguraci, která je ji předána a neobsahuje žádný stav kromě aktuální pozice kurzoru v editoru. Ačkoliv by bylo moudré, aby tento stav byl definován v nadřazené komponentě a následně byl pouze předán a zobrazen. Místo toho je definován zde a nastavení tohoto stavu je provedeno přes referenci z nadřazené komponenty, což je proti dobrým zásadám programování tak, jak bylo zmíněno v kapitole 4.2. V této situaci je problém, že pokud je vyvolána událost o změně kurzoru z komponenty editoru při editaci textu doprovázena událostí o změně obsahu, vznikne požadavek na dvojí zpracování, provede se zápis pozice kurzoru, přepíše se komponenta a obsah, který uživatel vložil bude ztracen. Proto, když je tento stav oddělen, může se pozice libovolně měnit bez ohledu na jiné události, protože stav je ovlivněn na jiných místech aplikace.
- **ToolList** – V této části jsou použité nástroje namapovány do řaditelného seznamu, včetně popisu jejich konfigurace a akcí, které s nimi lze provádět. Zde je vhodné zmínit způsob generování položek, protože akce dostupné přes tlačítko vlastností (angl. „properties“) potřebují vrchol, odkud se menu otevře. Zde však vzniká problém, protože list je generovaný během vykreslování a vrchol je nutné znát ještě před jeho započetím. Důsledkem pak je situace, kdy se všechna menu vykreslují v levém horním rohu monitoru (souřadnice `0,0`). Tento problém řeší externí knihovna **material-ui-popup-state**[1], která nabízí vlastní hooky, které sledují stav těchto rolovacích elementů a připojují k nim spouštěcí funkce, které pak nasměrují vykreslené položky menu na správné místo.
- **EditDialog** – Je dialogové okno dostupné přes seznam použitých nástrojů, které umožňuje zpětně editovat konfiguraci nástroje. V tomhle souboru je definován pouze abstrakt tohoto okna, který je společný pro všechny nástroje (popis, dialogové akce

[1] material-ui-popup-state: https://github.com/jcoreio/material-ui-popup-state

aj.). Samotný obsah je pak definován odpovídajícím nástrojem, který musí implementovat metodu `handleUpdate()`, která validuje platnost vstupů před potvrzením změn a následně uzavře okno. Obsah je pro každý nástroj individuální a tyto soubory jsou umístěny v adresáři `/src/components/edits/tools`. Kvůli většímu počtu těchto souborů jsou na obrázku 5.2 tyto komponenty nahrazeny textem.

- **ToolTabs** – Zde jsou definovány kategorie a k nim připojeny příslušné karty s nástroji. Jde spíše o abstrakt zajišťující navigaci a přizpůsobení šířce monitoru pomocí `MeadiaQuery` stejně jako v seznamu použitých nástrojů. Samotné karty s kategoriemi potom importují nástroje z adresáře `/src/components/tabs/tools`, kde opět na obrázku 5.2 jsou podmnožiny těchto nástrojů nahrazeny textem kvůli jejich počtu. Tato skutečnost umožňuje v budoucnu pak snadno zaměňovat obsah jednotlivých kategorií editací jednoho řádku.
- **FindTab** – Obsahuje nástroje, jejichž primárním účelem je hledání nebo porovnávání.
- **ReplaceTab** – Poskytuje nástroje pro náhradu textu.
- **ReduceTab** – Zde jsou nástroje, jejichž účelem je redukce textu.
- **ModifyTab** – Obsahuje nástroje, které nějakým způsobem zaměňují nebo obohacují text.
- **ToolHeader** – Je komponenta použita při definici každého nástroje a poskytuje její název a funkční popis. Výsledkem je pak jednotný styl pro všechny nástroje a snížena redundance kódu.
- **Alerts** – Zde jsou implementovány informační vyskakovací okna, která se zobrazují vlevo dole na monitoru po dobu 2 sekund. Jde primárně o sdělující animaci, takže spouštěcí akce je provedena přes referenci. Zpracování požadavku na zobrazení sdělení je zařazeno na konec fronty a obsluha je pak provedena následovně:
  - Je-li fronta prázdná zobrazí se sdělení ihned.
  - Pokud je sdělení aktuálně zobrazeno, zavře se a zobrazí se následující.
  - Není-li zobrazeno žádné sdělení a fronta je neprázdná, zařadí se požadavek na její konec.

## 5.3 Proces zpracování textu

Data potřebná ke zpracování včetně samotného procesu se nachází v komponentě na nejvyšší úrovni a transformaci je potřeba provést automaticky vždy, když uživatel provede akci, která ovlivňuje výsledek. Jedná se tedy o následující události:

- Změní se obsah editoru (se zpožděním 1,5 sekundy kvůli optimalizaci výkonu)
- Přepnutí režimu editoru
- Manipulace s breakpointy
- Přidání nástroje
- Editace nástroje

- Smazání nástroje
- Aktivace nebo deaktivace nástroje
- Změna pořadí nástrojů

```
1  useEffect(() => {
2      const worker = new WebWorker(pipeWorker); // Vytvoreni web workera
3
4      worker.postMessage({ // Odeslani zpravy s daty ke zpracovani
5          text: editorContent,
6          pipeline: pipeline,
7          breakpoints: inspectMode.breakpoints,
8          inspectMode: inspectMode.enabled
9      });
10
11     worker.onmessage = (event) => { // Definice reakci na odpovedi
12         if (event.data.type === "progress") {
13             setPipeProgress(event.data.data); // Zobrazeni postupu zpracovani
14         }
15         else {
16             setEditorResult(event.data.data); // Vypis vysledku
17         }
18     };
19
20     return () => { // Uklid
21         worker.terminate(); // Ukonceni workera
22     };
23 }, [editorContent, pipeline, inspectMode]);
```

Výpis 5.1: Komunikační cyklus zpracování textu. Na začátku je vytvořena instance web workera, kterému je zaslána zpráva s potřebnými daty ke zpracování. Následně je definované chování na příchozí odpovědi. V poslední fázi je provedeno jeho ukončení.

Klíčová část kódu pro zpracování textu, která pracuje s web workerem je zobrazena na výpisu 5.1. Zpracování se provádí v metodě `useEffect()`, která je podrobněji popsána v kapitole 4.2. Dojde-li ke změně stavu, není změna hned dostupná, ale stav se změní asynchronně později, protože React v daný moment ještě chvíli vyčkává na další případné změny stavů a poté všechny tyto změny provede naráz. Tato metoda je tedy zavolána potom, co už jsou tyto změny provedeny a platné.

Přesněji stavy, které jsou sledovány touto metodou jsou definovány na řádku 23 a jedná se o vstup editoru, použité nástroje a strukturu s ladícími daty. Jako první se inicializuje web worker a je mu zaslána zpráva s potřebnými daty ke zpracování textu. Tedy jedná se o řetězec vstupního textu, list s konfigurací použitých nástrojů, list breakpointů a boolovská hodnota stavu ladícího režimu. Následně na řádku 11 je definovaná reakce na odpovědi, kde se buď zobrazuje postup během procesu zpracování nebo se vypíše výsledek. Na závěr na řádku 20 je provedeno uvolnění alokovaných zdrojů, které se provede při odpojení (poslední fázi), aby na pozadí neběžely předešlé skripty workerů. Celý proces se pak při novém požadavku na zpracování opakuje.

Logika zpracování textu ze strany workera po přijetí požadavku s potřebnými parametry je znázorněna v algoritmu 1. V případě ladícího režimu, kde je požadována transformace typicky nad malém množství záznamů, je transformace provedena také nad celými daty

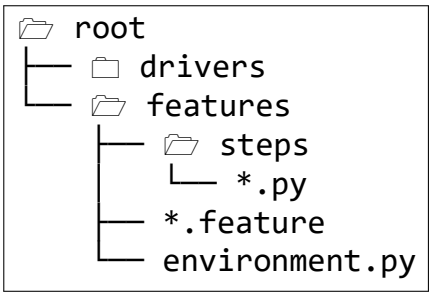
a poté jsou tyto požadované části vyselektovány, jelikož některé nástroje vyžadují znalost okolních záznamů (kontext) a jen aktuální záznam nestačí. Zároveň některé nástroje zaměňují jejich pořadí či je jinak modifikují, proto je nutné každý záznam očíslovat, aby byl nakonec dohledatelný. Na začátku algoritmu je provedeno zpracování vstupního textu do vhodné podoby. V případě ladícího režimu je text uložen do struktury, kde každý záznam obsahuje data a právě dané číslo. V běžném režimu postačí text jako klasický řetězec. Dále je definován přírůstek na ukazateli průběhu, který bude odeslán po zpracování každého použitého nástroje. Následuje zpracování všech aktivních nástrojů postupně za sebou nad celým textem. Výsledek z každého nástroje je pak předán následujícímu. Zde je opět nutné rozlišit způsob zpracování, které se buď provádí nad řetězcem nebo nad strukturou. V případě dat uložených ve struktuře je záznam, který není obsažen ve výsledku označen klíčovým slovem `null`. Po zpracování celého textu je výsledek odeslán zpět do aplikace. V případě ladícího režimu je nutné ještě předem výsledná data ze struktury extrahovat pomocí čísel řádků, nad kterými jsou nastavené breakpointy a vytvořit z nich řetězec.

**Algorithm 1** Proces zpracování textu

```
1:  procedure ProcessPipeline(text, pipeline, breakpoints, inspectMode)
2:      if inspectMode == false then                          ▷ Načtení textu
3:          data ← text
4:      else                                  ▷ Označení řádků pomocí struktury
5:          i ← 0
6:          for each line ∈ text do
7:              data[i] ← {number: i + 1, data: line}
8:              i ← i + 1
9:      unit ← 100 / len(pipeline)
10:     if len(pipeline) > 0 then
11:         postMessage({type: "progress", data: 0})   ▷ Informace o procesu zpracování
12:     i ← 0
13:     for each tool ∈ pipeline do                          ▷ Zpracování nástrojů
14:         if tool.active() == true then
15:             if inspectMode == false then
16:                 data ← processTool(data, tool)
17:             else
18:                 data ← processInspectTool(data, tool)
19:         postMessage({type: "progress", data: (i + 1) ∗ unit})
20:         i ← i + 1
21:     if inspectMode == true then                  ▷ Extrakce dat ze struktury
22:         temp ← empty list
23:         for each line ∈ data do
24:             if breakpoints.includes(line.number) && line.data != null then
25:                 temp.append(line.data)
26:         data ← temp.join('\n')
27:     postMessage({type: "result", data: data})              ▷ Odeslání výsledku
```

## 5.4 Tvorba automatizovaných testů

Testy byly vytvořeny samostatně od hlavního projektu a byly implementovány v Pythonu 3.6 na frameworku Selenium (viz kapitola 4.6) s BDD scénáři (viz kapitola 6.2). Struktura adresáře testů je znázorněna na obrázku 5.3, kde v adresáři `drivers` jsou umístěny potřebné ovladače k ovládání prohlížeče. Při testování byl použit ovladač pro Firefox GeckoDriver[2]. V adresáři `features` jsou napsány všechny testovací scénáře, jejichž kroky jsou pak implementovány v adresáři `features/steps`. Důležitým souborem je soubor `environment.py`, ve kterém je provedeno nastavení prostředí před spuštěním testů (volba prohlížeče, ovladače, doba čekání atd.) a úklid po jejich provedení.




Obrázek 5.3: Adresářová struktura testů.

```
1  Feature: Editor
2
3      Scenario: Krok zpet
4          Given Uzivatel se nachazi na strance weboveho nastroje
5          And Do vstupniho editoru je vlozeno "text"
6          When Uzivatel vrati krok zpet
7          Then Vstupni editor obsahuje ""
8
9      Scenario: Krok dopredu
10         Given Uzivatel se nachazi na strance weboveho nastroje
11         And Do vstupniho editoru je vlozeno "text"
12         When Uzivatel vrati krok zpet
13         And Uzivatel se vrati krokem dopredu
14         Then Vstupni editor obsahuje "text"
```

Výpis 5.2: Testovací scénáře v jazyce Gherkin.

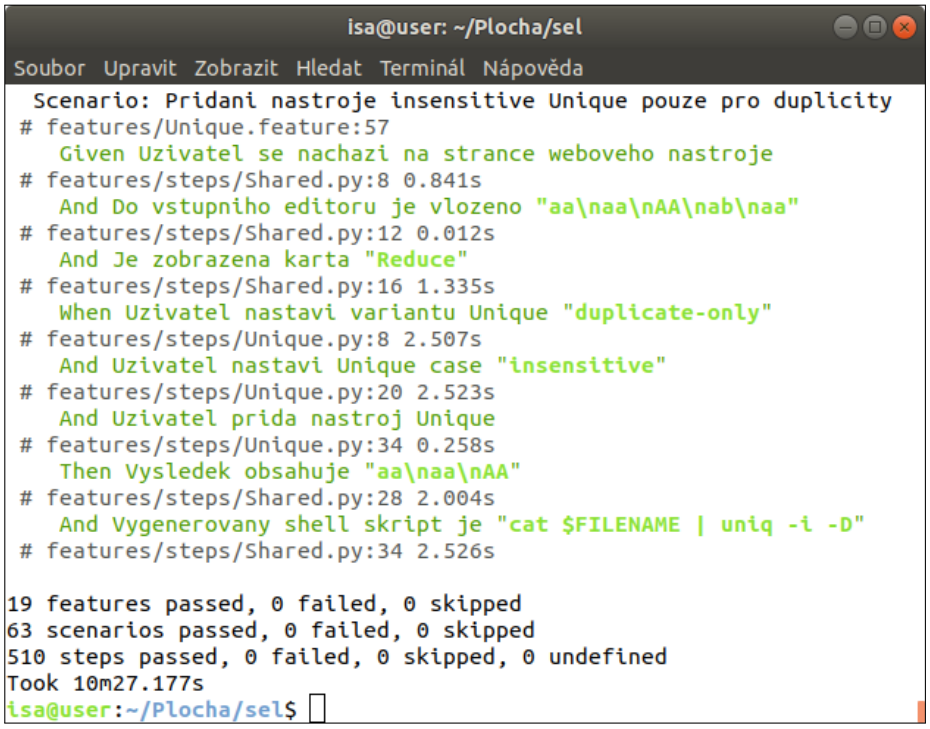
Příklad testovacího scénáře je zobrazen na výpisu 5.2, kde:

- `Feature` – Je abstraktní popis následující skupiny scénářů.
- `Scenario` – Obsahuje název testovacího případu.
- `Given` – Definuje počáteční kontext před provedením testu, tedy to, co musí být nastaveno před zahájením.
- `When` – Jsou kroky popisující události provedeny interakcí uživatele s aplikací.
- `Then` – Popisuje očekávaný výsledek.

[2] GeckoDriver: https://github.com/mozilla/geckodriver/releases

- `And` – Je klíčové slovo, které dědí funkcionalitu předešlého pro plynulé strukturovaní. Např. při použití více akcí `Then` jej definujeme pouze jednou a následně používáme slovo `And`.

Pro automatizaci běhů testů je použit program Behave[3], který lze spustit příkazem `behave` v kořenovém adresáři projektu po správné konfiguraci potřebných závislostí. Program poté začne vyhledávat `.feature` soubory s testovacími scénáři v adresáři `features` a jejich implementované kroky v adresáři `features/steps`, proto je nutné zachovat tuto adresářovou strukturu. Výpis výsledků je pak zobrazen na obrázku 5.4.




Obrázek 5.4: Výsledky implementovaných testů.

## 5.5 Nasazení aplikace

Aplikace[4] byla zveřejněna pomocí služby GitHub Pages (viz kapitola 4.7), kvůli které byla přidána další knihovna `gh-pages`[5], která usnadňuje a automatizuje proces nasazení. Nasazení sestavené aplikace je provedeno ve zvláštní větvi GitHub repozitáře projektu pro gh-pages, odkud bude aplikace dostupná. Samotná konfigurace skriptů pro nasazení je znázorněna na výpisu 5.3 a nachází se v souboru `package.json`. Nasazení lze pak provést přes terminál v kořenovém adresáři projektu příkazem `npm run deploy`, který vloží sestavenou aplikaci na web z adresáře `build`. Ovšem před provedením této operace je automaticky spuštěn příkaz `predeploy`, který obsahuje skript `npm run build` pro sestavení aplikace. Odkaz, přes který bude aplikace dostupná je definován na řádku 1 (viz výpis 5.3), jinak by se aplikace nezobrazovala korektně, přesněji by se zobrazilo pouze bíle okno.

[3]Behave: https://behave.readthedocs.io/en/latest
[4]Craftex: https://craftex.app
[5]gh-pages: https://www.npmjs.com/package/gh-pages

```
"homepage": "https://craftex.app",
"scripts": {
    "predeploy": "npm run build",
    "deploy": "gh-pages -d build",
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
},
```

Výpis 5.3: Konfigurace skriptů

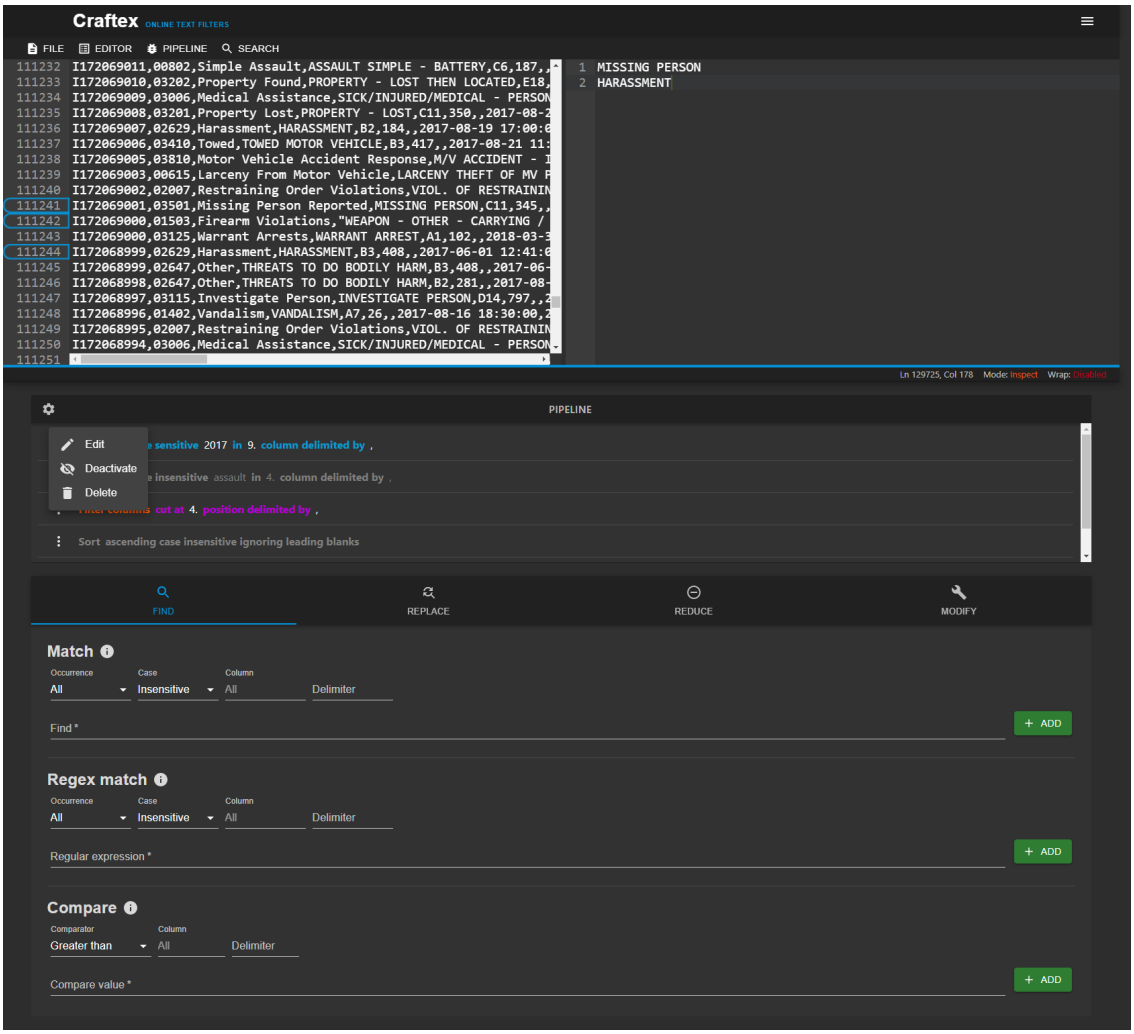
Vždy před nasazením aplikace byla provedena bezpečnostní revize příkazem `npm audit`, který hledá zranitelnosti projektu ve stromu závislostí. Pokud jsou detekovány nějaké chyby, lze je někdy opravit automaticky příkazem `npm audit fix` (případně opakovaně, protože opravení některých chyb může napomoci opravení jiných, dříve automaticky neopravitelných), v opačném případě je nutné provést opravu ručně.

Doména aplikace byla zvolena s příponou `.app` za cenu 435 Kč/rok, která je určena především pro aplikace a vyžaduje aplikační protokol HTTPS, který zároveň služba GitHub Pages podporuje. Kromě nastavení DNS záznamů je nutné do kořenového adresáře sestavené aplikace umístit soubor s názvem `CNAME` obsahující doménu aplikace, aby se GitHub mohl na aplikaci pomocí této domény odkázat.

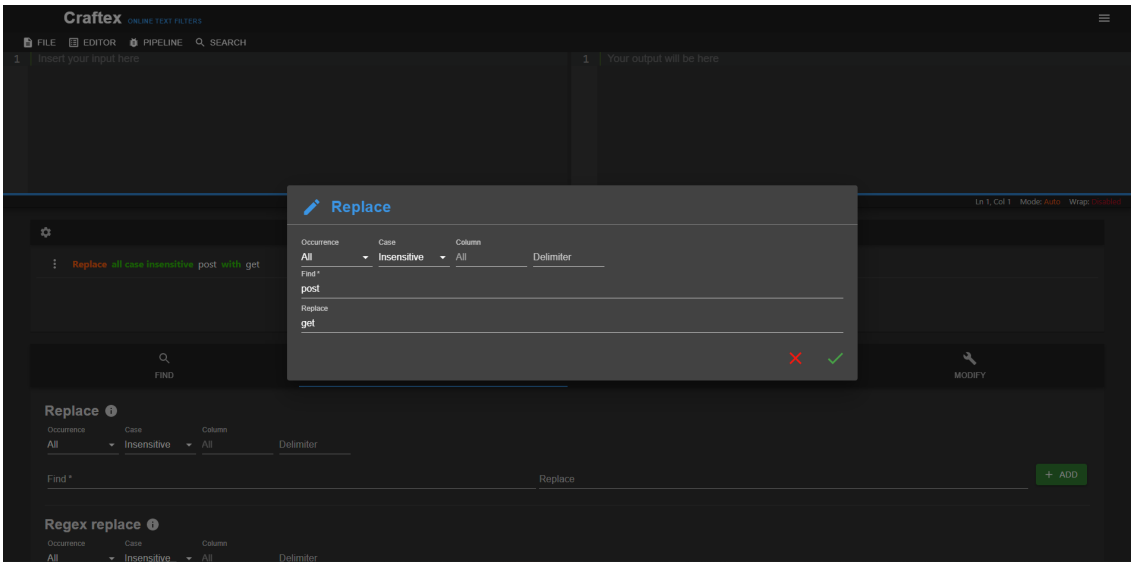
## 5.6 Výsledná podoba aplikace

Na obrázku 5.5 je zobrazena výsledná podoba aplikace. V horní liště se nachází záložky s funkcemi rozdělené do kategorií pro soubory, editor, použité nástroje a vyhledávač, který je společný pro vstupní i výstupní editor. Taktéž je možno použít klávesovou zkratku „ctrl + F“ v každém editoru separátně. Velikost editorů lze měnit do výšky i do šířky a ladění je realizováno pomocí breakpointů (například řádek 111 241 na obrázku 5.5). Těsně pod editory se nachází modrá lišta zobrazující stav při převodu. Dále se pod ní nachází lišta s informacemi o aktuálním nastavení editoru. Zhruba uprostřed aplikace se nachází sekce pojmenovaná anglicky „pipeline“, která zobrazuje aktuálně použité nástroje. Každý nástroj je možné dodatečně editovat (viz dialogové okno na obrázku 5.6), deaktivovat nebo odstranit. Zpracování je prováděno shora dolů a pořadí je možné libovolně zaměňovat prostým přetažením. Velikost této sekce je taktéž možné měnit do výšky. Pomocí dialogového okna je také zobrazován shell skript, který je možné vidět na obrázku 5.7. V poslední části se nachází galerie nástrojů rozdělena do kategorií podle primárního účelu pro usnadnění orientace. V případě chybně zadaných parametrů je uživatel upozorněn hláškou a zvýrazněním příslušného vstupu. Každý nástroj obsahuje funkční popis, který slouží jako manuálová stránka. Celý návrh je responzivní a použitelný i na mobilních zařízeních (viz obrázek 5.8). Hlavní změnou na mobilním zařízení je eliminace odsazení od krajů prohlížeče a poskytnutí tak více prostoru pro vstupy, zároveň dialogová okna jsou zobrazována přes celé okno zařízení. Ostatní funkcionalita zůstává zachována a při použití zařízení na šířku je pak práce s aplikací z mého pohledu ještě jednodušší. Výslednou podobu lze také vidět na vytvořeném demonstračním videu[6] dostupném na internetovém portálu YouTube.

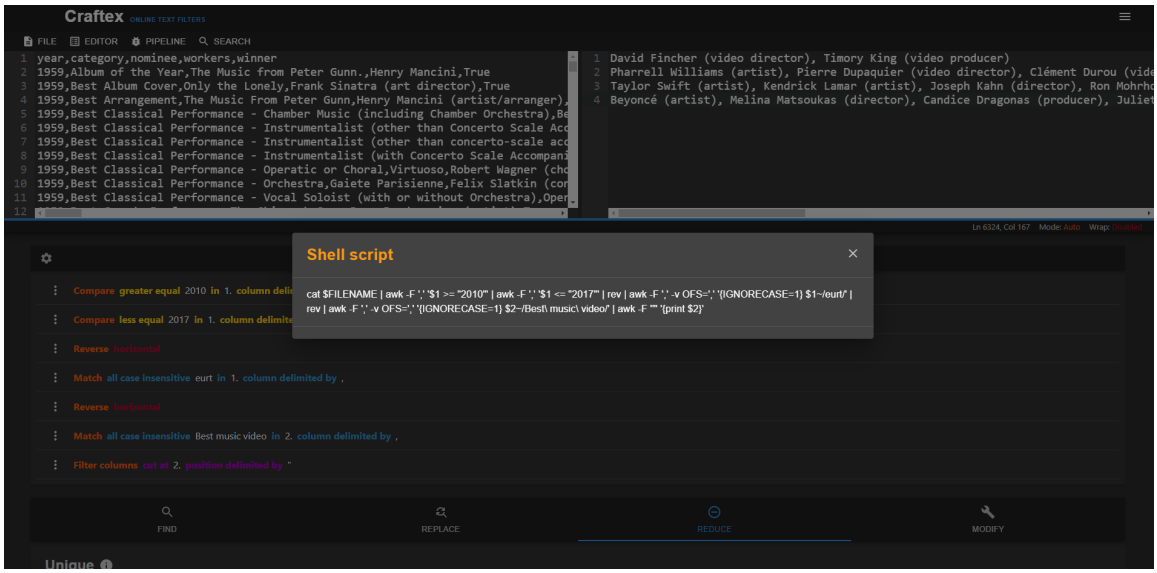
[6] Demo: https://youtu.be/KA4wxr-8vKk



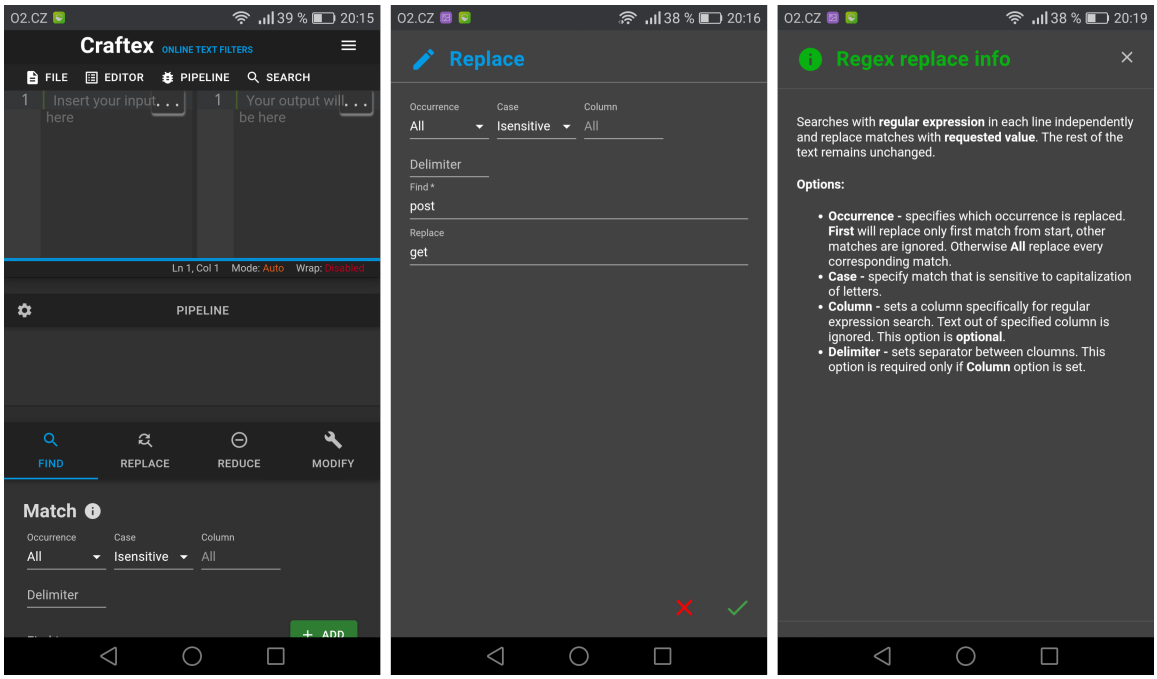

Obrázek 5.5: Výsledná podoba aplikace.




Obrázek 5.6: Editace použitého nástroje.




Obrázek 5.7: Zobrazení vygenerovaného shell skriptu.




Obrázek 5.8: Aplikace na mobilním zařízení. V levé části je zobrazena hlavní stránka aplikace, uprostřed se nachází editace nástroje a vpravo je zobrazen funkční popis nástroje.

# Kapitola 6

# Sběr dat a testování

V této kapitole je zmíněna testovací datová sada a hlavní zdroje, ze kterých byla pořízena, včetně ukázek její filtrace. Další část je věnována metodice provedeného automatizovaného testování. Poslední část obsahuje uživatelské testování, kde se uživatelé podíleli svoji zpětnou vazbou na doladění aplikace a je také zmíněn výsledek provedeného časového testu, jehož cílem bylo zhodnotit efektivitu výsledného řešení.

## 6.1 Testovací datová sada

Během procesu implementace byla shromažďována reálná vstupní data vhodná k filtraci, která byla použita zejména při uživatelském testování a prezentaci aplikace. Nejčastěji byla data získávaná od ostatních uživatelů, z různých míst operačního systému nebo z následujících portálů:

- **Kaggle**[1] – Dceřiná společnost společnosti Google, která poskytuje datové sady z různých oborů vědy a od odborníků na strojové učení.
- **DataHub**[2] – Je portál poskytující kolekce dat z různých zdrojů ve formě architektury hub.
- **Data.gov**[3] – Je americká vládní stránka poskytující hodnotné informace z federální, vládní a státní oblasti pro širokou veřejnost.

Celkem bylo nasbíráno přes 60 různých exemplářů s celkovou velikostí přes 200 MB. Možné případy takto vyfiltrovaných dat jsou znázorněny níže.

### 1. ukázka

Data byla pořízena ze školního serveru Merlin, ze kterých jsou vyfiltrováni uživatelé, kteří mají spuštěnou instanci textového editoru `vim`, včetně časového údaje, kdy byl proces spuštěn. Filtrace je znázorněna na obrázku 6.1, fragment vstupních dat na výpisu 6.1 a fragment výstupních dat na výpisu 6.2.

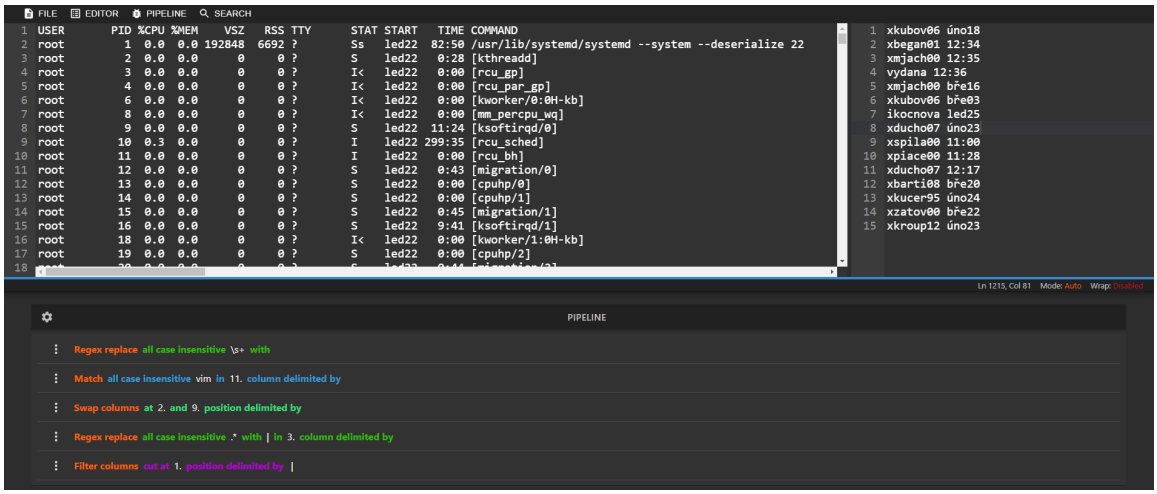
[1] Kaggle: https://www.kaggle.com
[2] DataHub: https://datahub.io
[3] Data.gov: https://www.data.gov

```
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
root 1 0.0 0.0 192848 6692 ? Ss led22 82:50 /usr/lib/systemd/systemd --system
    --deserialize 22
root 2 0.0 0.0 0 0 ? S led22 0:28 [kthreadd]
```

Výpis 6.1: Fragment vstupních dat.

```
xkubov06 úno18
xbegan01 12:34
xmjach00 12:35
```

Výpis 6.2: Fragment výstupních dat.




Obrázek 6.1: Filtrace uživatelů s aktivní instancí textového editoru `vim` s časovým údajem.

### Vygenerovaný ekvivalentní shell skript

```
cat $FILENAME | sed -E 's/\s+/\ /gI' | awk -F '[ ]' -v OFS=' '
'{IGNORECASE=1} $11~/vim/' | awk -F '[ ]' -v OFS=' '
'{t=$2;$2=$9;$9=t;print;}' | awk -F '[ ]' -v OFS=' '
'{IGNORECASE=1}{gsub(".*", "|", $3); print }' | awk -F '|' '{print $1}'
```

### 2. ukázka

V tomto případě (viz obrázek 6.2) byla data filtrována nad log souborem generovaný ovladačem prohlížeče při automatickém testování (viz fragment na výpisu 6.3). Z dat byly získány všechny unikátní chybové záznamy (viz fragment na výpisu 6.4).

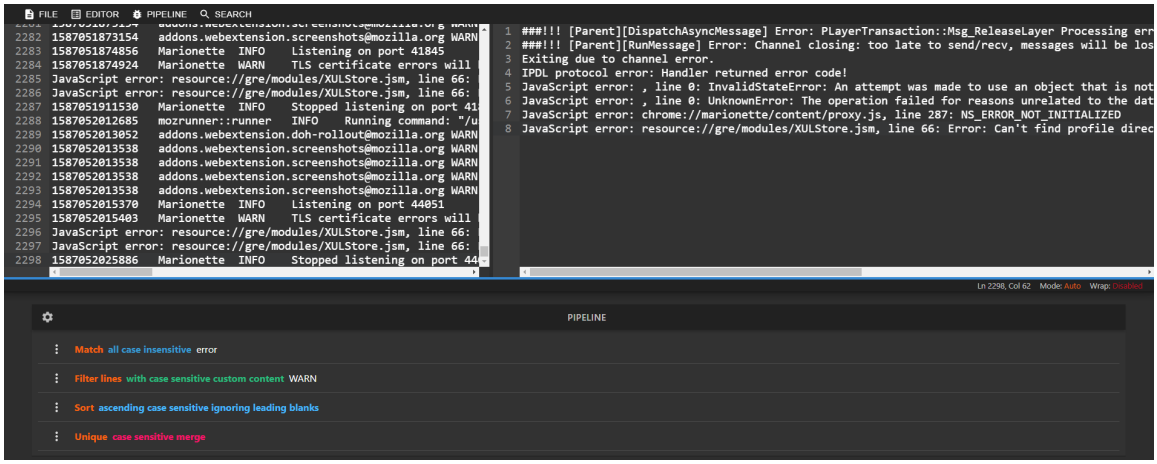
```
1586875405783 mozrunner::runner INFO Running command: "/usr/bin/firefox"
    "-marionette" "-foreground" "-no-remote" "-profile"
    "/tmp/rust_mozprofileuOpWOb"
1586875406178 addons.webextension.doh-rollout@mozilla.org WARN Loading extension
    'doh-rollout@mozilla.org': Reading manifest: Invalid extension permission:
    networkStatus
```

```
3 1586875406681 addons.webextension.screenshots@mozilla.org WARN Loading extension
    'screenshots@mozilla.org': Reading manifest: Invalid extension permission:
    mozillaAddons
```

Výpis 6.3: Fragment vstupních dat.

```
1 ###!!! [Parent][DispatchAsyncMessage] Error: PLayerTransaction::Msg_ReleaseLayer
    Processing error: message was deserialized, but the handler returned false
    (indicating failure)
2 ###!!! [Parent][RunMessage] Error: Channel closing: too late to send/recv,
    messages will be lost
3 Exiting due to channel error.
```

Výpis 6.4: Fragment výstupních dat.




Obrázek 6.2: Filtrace chybových záznamů z webového ovladače Gecko.

### Vygenerovaný ekvivalentní shell skript

```
cat $FILENAME | grep -E -i 'error' | sed -E '/WARN/d' |
LC_ALL=C sort -sb | uniq
```

## 6.2 Automatizované testování

Kvůli častým změnám v kódu nebo aktualizaci stromu závislostí byly implementovány automatizované testy, jejichž účelem bylo otestovat základní funkcionalitu aplikace a případně upozornit na chyby, které by mohly vzniknout při nekompatibilitě. Výsledkem bylo ušetření velkého množství času, který by musel být vynaložen při manuálním testovaní mnou nebo jinými testery při opakujících se operacích s každou novou verzí aplikace.

Testy [6] byly napsány na metodice **behavior-driven development** (dále jen BDD), což umožňuje psát nebo číst průběh testů v přirozeném jazyce i uživateli, který nemusí disponovat znalostmi programování (např. analytik, stakeholder nebo koncový uživatel) a tester poté tyto kroky implementuje. Testy jsou děleny na testovací případy, které se

nazývají scénáře a jsou napsány v jazyce Gherkin[4], který je strukturován v pořadí kontext-akce-výsledek. Jeho účelem je srozumitelně popsat jakým způsobem by se měl daný objekt chovat v různých situacích s různými parametry. Tento scénář je pak uložen v souboru se speciální formátem `.feature`, který je poté čitelný nástroji podporující BDD jako např. Cucumber, JBehave, Behat aj. Celkem bylo implementováno přes 500 kroků ve více než 60 scénářích.

## 6.3 Uživatelské testování

Během implementace probíhala konzultace se čtyřmi potenciálními uživateli aplikace (s dalšími jen příležitostně), se kterými byla validována přehlednost uživatelského rozhraní. Uživatelé byli vybráni podle následujících kategorií:

- **Technicky zdatný** – Celkem 1 uživatel.
- **Technicky středně zdatný** – Celkem 2 uživatelé.
- **Běžný uživatel** – Celkem 1 uživatel.

Testování odhalilo několik chyb (bugů), které byly opraveny. Zejména se jednalo o operace nad neočekávanými daty nebo o problémy s dynamikou webu, kdy se při určitém pořadí změn velikostí elementů měnila šířka obou editorů nebo po změně velikosti editoru a následným změnám okna prohlížeče, zůstala pozice výstupního editoru statická, přestože by měl poměr zůstat zachovaný. Zároveň také vzniklo několik dalších požadavků a nápadů, které byly z velké části do aplikace ať už přímo nebo nepřímo implementovány. Takovým nápadem byl například popis funkčnosti nástrojů, společný vyhledávač, aj.

Zajímavým poznatkem z testování byla skutečnost, že běžní uživatelé, kteří podobné nástroje neznají nebo nepoužívají, provádějí filtraci nad jednoduššími daty a používají poměrně málo operací, naopak pokročilejší uživatelé s odborněji zaměřenými daty provádějí více operací a jejich požadavky na výstupy jsou velmi konkrétní. Největší problém v orientaci bylo dělení nástrojů do kategorií. Každý uživatel měl jiný názor na jejich dekompozici ať už do počtu kategorií nebo jejich členění či jinak, bylo proto nutné najít optimální střední cestu, která každého zčásti uspokojila, přestože obsahovala pár kompromisů. Výsledný návrh poté neprokázal chaotické řešení a uživatelé po rychlém prozkoumání těchto kategorií byli schopni plynule pracovat. Teorie k názvům je potom následující:

- **Find** – Obsahuje nástroje, které vyberou záznamy podle zvolených kritérií. O výsledných záznamech pak lze s jistotou říci, že obsahují právě tuto informaci.
- **Replace** – Nástroje v této kategorii jsou určeny pro náhradu části vstupního textu.
- **Reduce** – V této kategorii jsou obsažený nástroje, které nějakým způsobem text redukují. O výsledných záznamech nemusíme mít konkrétní informaci, kterou nesou.
- **Modify** – Zde jsou nástroje, které žádným způsobem text neredukují, ale modifikují nebo obohacují.

[4]Gherkin: https://cucumber.io/docs/gherkin/reference

### Měřený úkol

Cílem poslední části testování bylo zjistit, s jakou časovou efektivitou je uživatel schopen data vyfiltrovat oproti klasickému linuxovému skriptování v shellu. Technicky zdatnému uživateli, jehož náplní práce není přímo tato problematika, přesto mu není úplně cizí, byla poskytnuta data, která musel následně transformovat do požadované podoby v aplikaci (viz obrázek 6.3), přičemž mu byl měřen čas. Poté jeho úkolem bylo provést tu stejnou transformaci nad stejnými daty v shellu pomocí vyhledávače na internetu. Výsledné časy jsou zaznamenány v tabulce 6.1.

### Zadání

Byl poskytnutý přístupový log soubor ze serveru, který obsahoval přes 21 000 záznamů (viz fragment na výpisu 6.5). Úkolem bylo vypsat všechny IP adresy, které poslali požadavek na server po datu 27.12.2015 (včetně), seznam adres musel být bez redundanci s prefixem počtu přístupu dané adresy (viz fragment na výpisu 6.6).

```
1 109.169.248.247 - - [12/Dec/2015:18:25:11 +0100] "GET /administrator/ HTTP/1.1"
    200 4263 "-" "Mozilla/5.0 (Windows NT 6.0; rv:34.0) Gecko/20100101
    Firefox/34.0" "-"
2 109.169.248.247 - - [12/Dec/2015:18:25:11 +0100] "POST /administrator/index.php
    HTTP/1.1" 200 4494 "http://almhuette-raith.at/administrator/" "Mozilla/5.0
    (Windows NT 6.0; rv:34.0) Gecko/20100101 Firefox/34.0" "-"
3 46.72.177.4 - - [12/Dec/2015:18:31:08 +0100] "GET /administrator/ HTTP/1.1" 200
    4263 "-" "Mozilla/5.0 (Windows NT 6.0; rv:34.0) Gecko/20100101 Firefox/34.0"
    "-"
```

Výpis 6.5: Fragment výstupních dat.

```
1 5 1.0.188.61
2 1 101.226.169.219
3 3 104.144.248.16
```

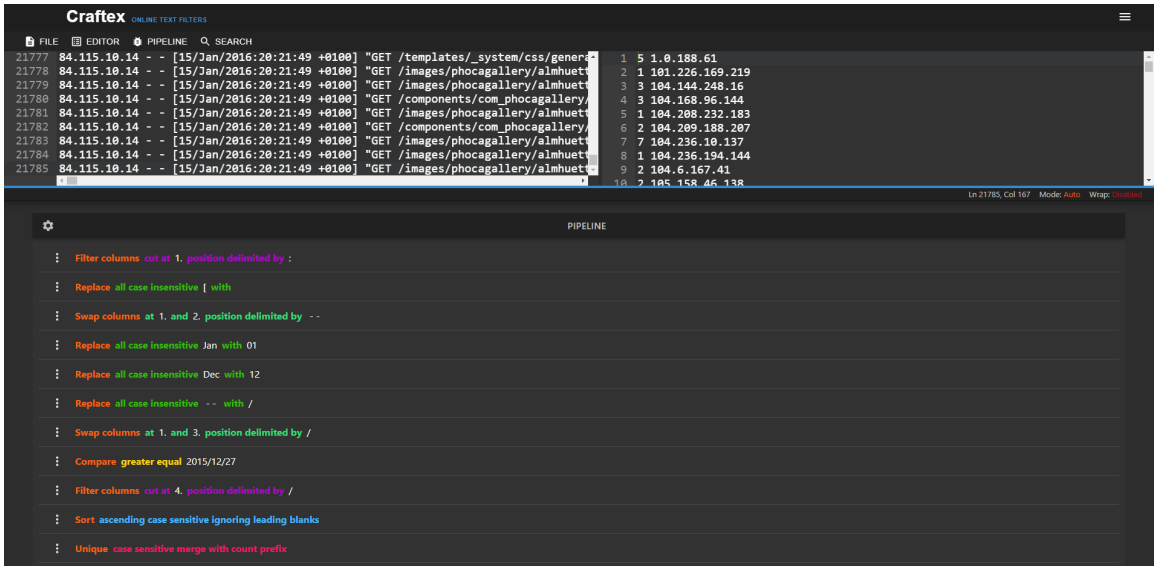
Výpis 6.6: Fragment výstupních dat.

### Vygenerovaný ekvivalentní shell skript

```
cat $FILENAME | awk -F ':' '{print $1}' | sed -E 's/\[//gI' |
awk -F ' - - ' -v OFS=' - - ' '{t=$1;$1=$2;$2=t;print;}' |
sed -E 's/Jan/01/gI' | sed -E 's/Dec/12/gI' | sed -E 's/\ \-\ \-\ /\//gI' |
awk -F '/' -v OFS='/' '{t=$1;$1=$3;$3=t;print;}' |
awk -F '\n' '$1 >= "2015/12/27"' | awk -F '/' '{print $4}' |
LC_ALL=C sort -sb | uniq -c | sed 's/^[[:blank:]]*//'
```

Tabulka 6.1: Výsledné časy při testování

| | **Aplikace** | **Shell** |
|---|---|---|
| **Čas** | 11 minut 36 sekund | 1 hodina 6 minut 25 sekund |




Obrázek 6.3: Výpis IP adres včetně počtu jejich přístupů na server po datu 27.12.2015 (včetně) z přístupového log souboru.

### Doplňující otázky

Po provedení úkolu byly uživateli položeny následující otázky, jejichž účelem je zhodnotit přínos aplikace a poukázat na její výhody či nedostatky.

- *Narazil jste při práci na nějaké problémy? Případně změnil byste něco?* Uspořádání nástrojů by mohlo být podle abecedy, možná bych to pak vyhledal rychleji.
- *Kdyby jste dostal podobné zadání na zpracování prostého textu, jak by jste tento problém řešil?* Nejspíše bych použil Excel, PlanMaker nebo případně bych data manuálně zpracoval, pokud by nebyla tak rozsáhlá.
- *Čím Vás aplikace oslovila?* Přátelské rozhraní (hlavně vůči terminálu), data lze zpracovat bez znalostí funkcí oproti Excelu. Aplikaci bych i znovu použil, ale záleží na objemu dat, jejich formátu a operacích.
- *Myslíte, že je v aplikaci něco zbytečného nebo nadbytečného?* Nic mě nenapadá, dokážu si představit situace, ve kterých má každá funkcionalita svůj účel.
- *Myslíte, že v aplikaci něco schází?* Vždycky se dá něco vylepšit, teď by se mi zrovna hodilo řazení měsíců v roce podle jejich názvu místo číselné hodnoty.

# Kapitola 7

# Závěr

Cílem této práce bylo navrhnout a implementovat webový nástroj pro filtraci textu, který umožní uživatelům snadno a efektivně zpracovat data s možností prototypování a ladění, který řeší danou problematiku jinak než doposud jiné internetové aplikace. Důležitou částí bylo aplikaci testovat nad vhodnými daty, prokázat tak její funkčnost a odladit ji co nejvíce pro koncové uživatele.

V teoretické části byla nastíněna problematika filtrace, kde je možné se s ní setkat a byla provedena analýza dosavadních řešení. Následně bylo navrhnuto řešení této práce a návrh rozhraní za pomoci interaktivní makety. Po volbě použitých technologií, zvážení alternativ a následné implementaci vznikla funkční aplikace, která je schopna text plynule převádět, generovat nápomocné skripty a nabízí nemalé množství nástrojů. Hlavním úspěchem je snadné ladění, které je realizováno na více místech aplikace, zejména v oblasti pipeline dodatečnou manipulací, v oblasti textu pomocí breakpointů nebo jiných funkcí, které aplikace nabízí. Nemalá část této práce byla věnována testování, během kterého byla pořízena testovací datová sada. V této fázi byly implementovány automatizované testy a provedeno uživatelské testování, které se promítlo i do výsledné podoby aplikace.

Pro budoucí rozšíření je možné aplikaci obohatit o serverovou část, která umožní uživatelům zakládat účty, pod kterými budou moci své filtry ukládat a sdílet pomocí odkazů. Další možností může být zavedení tipů pro zefektivňování řazení nástrojů (např. 2× stejné otočení nemá na výsledek žádný efekt apod.).

Osobně mě práce obohatila o zkušenosti všemi fázemi vývoje aplikace od návrhu, přes implementaci až po testování a nasazení. Zejména část testování byla časově náročnější, než jsem původně očekával a tuto znalost určitě uplatním v budoucích projektech.

# Literatura


[1] Banks, A. a Porcello, E. *Learning React: Functional Web Development with React and Redux*. 1. vyd. Sebastopol: O'Reilly Media, květen 2017. 350 s. ISBN 9781491954621.

[2] Dawson, C. *JavaScript's History and How it Led To ReactJS* [online]. 2014. Aktualizováno 25. 7. 2014 [cit. 2. dubna 2020]. Dostupné z: https://thenewstack.io/javascripts-history-and-how-it-led-to-reactjs.

[3] Deshpande, C. *What is Selenium?* [online]. 2020. Aktualizováno 27. 2. 2020 [cit. 20. dubna 2020]. Dostupné z: https://www.simplilearn.com/what-is-selenium-article.

[4] Facebook. *Create a New React App* [online]. [cit. 2. dubna 2020]. Dostupné z: https://reactjs.org/docs/create-a-new-react-app.html.

[5] Gaunt, M. *Service Workers: an Introduction* [online]. 2019. Aktualizováno 9. 8. 2019 [cit. 6. dubna 2020]. Dostupné z: https://developers.google.com/web/fundamentals/primers/service-workers.

[6] Ghahrai, A. *BDD Guidelines and Best Practices* [online]. 2016. Aktualizováno 9. 11. 2016 [cit. 20. dubna 2020]. Dostupné z: https://devqa.io/agile/bdd-guidelines-best-practices.

[7] GitHub. *About GitHub Pages* [online]. [cit. 26. dubna 2020]. Dostupné z: https://help.github.com/en/github/working-with-github-pages/about-github-pages.

[8] Imran, D. *Everything React — All about React!* [online]. 2019. Aktualizováno 5. 2. 2019 [cit. 6. dubna 2020]. Dostupné z: https://blog.usejournal.com/everything-react-all-about-react-6d7a8de4bb05.

[9] Lardinois, F. Adobe launches Experience Design CC, a new tool for UX designers. *Techcrunch.com*. březen 2016.

[10] Mc, A. *What Does Create-React-App Actually Do?* [online]. 2019. Aktualizováno 7. 3. 2019 [cit. 2. dubna 2020]. Dostupné z: https://levelup.gitconnected.com/what-does-create-react-app-actually-do-73c899443d61.

[11] Mew, K. *Learning Material Design*. 1. vyd. Birmingham: Packt Publishing, prosinec 2015. 186 s. ISBN 9781785289811.

[12] Miller, J. *Threading the web with module workers* [online]. 2019. Aktualizováno 17. 12. 2019 [cit. 15. dubna 2020]. Dostupné z: https://web.dev/module-workers.

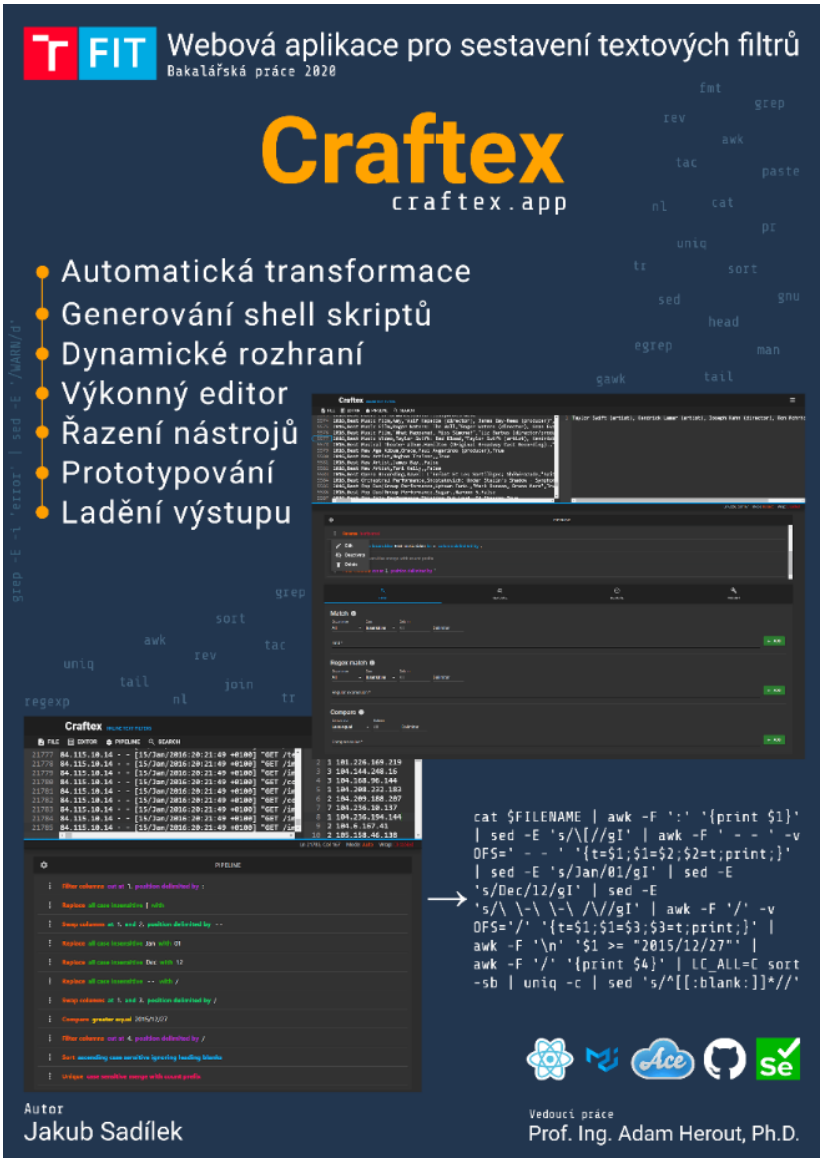
[13] Singh, A. Beginners Guide to ReactJS. *Medium.com.* květen 2019.

[14] Sumon, S. *Angular vs. React vs. Vue Detailed Comparison Guide - Which One to Choose in 2020* [online]. 2020. Aktualizováno 8. 4. 2020 [cit. 12. dubna 2020]. Dostupné z: https://www.themexpert.com/blog/angular-vs-react-vs-vue.

[15] Wanyoike, M. a Dierx, P. *A Beginner's Guide to npm, the Node Package Manager* [online]. 2020. Aktualizováno 9. 3. 2020 [cit. 1. dubna 2020]. Dostupné z: https://www.sitepoint.com/beginners-guide-node-package-manager.

# Příloha A

# Plakát

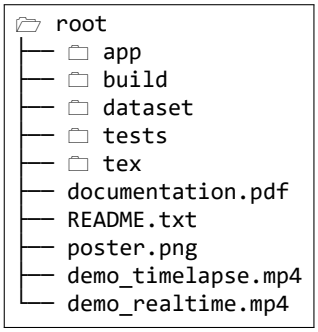



Obrázek A.1: Prezentační plakát webové aplikace ve snížené kvalitě.

# Příloha B

# Obsah přiloženého paměťového média

V této příloze je popsán obsah přiloženého paměťového média, který je zobrazen na obrázku B.1.




Obrázek B.1: Obsah paměťového média.

- **app** – Adresář obsahuje zdrojové kódy implementované aplikace.
- **build** – Adresář obsahuje zkompilované zdrojové kódy aplikace pro doménu `https://craftex.app`, proto je nelze odtud spustit a jsou zde uvedeny pouze pro úplnost.
- **dataset** – Adresář obsahuje datovou sadu s případy vhodnými k filtraci.
- **tests** – Adresář obsahuje zdrojové kódy automatizovaných testů.
- **tex** – Adresář obsahuje zdrojový tvar technické zprávy.
- **documentation.pdf** – Technická zpráva.
- **README.txt** – Manuál obsahující informace k instalaci a obsahu média.
- **poster.png** – Prezentační plakát aplikace v plné kvalitě.
- **demo_timelapse.mp4** – Zrychlené demonstrační video.
- **demo_realtime.mp4** – Demonstrační video v reálném čase.

# Příloha C

# Instalační manuál

V této příloze je popsán postup pro zprovoznění webové aplikace v prostředí Reactu a implementovaných testů na frameworku Selenium. Proces instalace byl testován na linuxové distribuci Ubuntu 20.04 LTS.

## C.1 Instalace aplikace

Následující kroky nainstalují potřebný software stejných verzí, které byly použity při implementaci, přestože mohou fungovat i jiné. Aplikace je také dostupná na webovém portálu https://craftex.app.

### Instalace Node.js verze v12.13.0

1. `sudo apt-get update`

Pro instalaci specifické verze Node.js je zapotřebí Node Version Manager (NVM).

2. `sudo apt install curl`
3. `curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.33.11/install.sh | bash`
4. Nyní je nutné zavřít a znovu otevřít terminál, tím bude povoleno NVM.

Kontrola nainstalované verze NVM.

5. `nvm --version`

Následujícím příkazem bude provedena instalace požadované verze Node.js (v12.13.0).

6. `nvm install 12.13.0`

Kontrola nainstalované aktuální verze Node.js.

7. `node -v`

   **(Poznámka)** V případě používané jiné verze je potřeba provést kontrolu již nainstalovaných verzí a vybrat v12.13.0 následujícími příkazy:

   `nvm ls`

   `nvm use 12.13.0`

### Instalace NPM verze 6.12.0

Po úspěšné instalaci Node.js verze v12.13.0, automaticky by měl být dostupný NPM verze 6.12.0, který je aktuálně vyžadován. Kontrolu nainstalované verze lze provést následujícím příkazem:

1. `npm -v`

   **(Poznámka)** V případě odlišné verze NPM lze provést instalaci požadované verze následujícím příkazem:

   `sudo npm install -g npm@6.12.0`

### Instalace aplikace

Otevřít kořenový adresář aplikace. V přiloženém paměťovém médiu se jedná o adresář `/app`.

1. `cd app`

Následujícím příkazem proběhne instalace závislostí aplikace.

2. `npm install`

   **(Poznámka)** Zobrazení zranitelností po instalaci není nutné věnovat pozornost, v době odevzdání byly v pořádku a přibývají poměrně pravidelně a spousta z nich jich je stále neopravena NPM. V následujícím odkaze si lze zobrazit přehled: https://www.npmjs.com/advisories.

Spuštění aplikace (může chvíli trvat).

3. `npm start`

   **(Poznámka)** V případě potíží je nutné použít právo root uživatele (`sudo npm start`).

Aplikace bude poté dostupná přes webový prohlížeč na adrese http://localhost:3000/. V tomto případě kód běžící na dev serveru není optimalizovaný a aplikace může být pomalejší. Pro vygenerování optimalizovaného kódu je potřebné spustit `npm run build`. Optimalizovaný kód bude poté umístěn v adresáři `./build`.

## C.2 Instalace testů

Otevřít kořenový adresář s testy. V přiloženém paměťovém médiu se jedná o adresář `/tests`.

1. `cd tests`

Pro běh je potřeba mít nainstalovaný python 3.6 a pip.

2. `sudo apt-get update`

3. `sudo apt-get install python3-pip`

Instalace Selenium a behave.

4. `pip3 install selenium`

5. `pip3 install behave`

6. `sudo apt install python3-behave`

Dále je potřeba zprovoznit webdriver pro ovládání prohlížeče (pro Firefox se jedná o GeckoDriver dostupny na: https://github.com/mozilla/geckodriver/releases), po stáhnutí je nutné ho umístit do adresáře `/usr/local/bin` a přidat právo pro spuštění (`chmod +x`), driver použitý při testováni je umístěn v adresáři `./drivers`, který lze také použít.

7. `sudo cp drivers/geckodriver /usr/local/bin`

Instalace GNU AWK (pro kompatibilitu s generovanými skripty).

8. `sudo apt-get update`
9. `sudo apt-get install gawk`

Spuštění testů nyní lze provést následujícími příkazy v kořenovém adresáři s testy (`/tests`).

- Spuštění všech testů: `behave`
- Spuštění feature: `behave -i <název>`
- Spuštění scénáře: `behave -n "<název>"`