VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

INTERAKTIVNÍ WEBOVÉ ROZHRANÍ PRO VÝBĚR VHODNÝCH ARCHITEKTUR

INTERACTIVE WEB APPLICATION FOR DESIGN SPACE EXPLORATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **DENIS SLÁVIK**
AUTHOR

VEDOUCÍ PRÁCE **Doc. Ing. ZDENĚK VAŠÍČEK, Ph.D.**
SUPERVISOR

BRNO 2018

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Slávik Denis**
Obor: Informační technologie
Téma: **Interaktivní webové rozhraní pro výběr vhodných architektur**
Interactive Web Application for Design Space Exploration
Kategorie: Web

Pokyny:

1. Seznamte se s problematikou vizualizace řešení s vícedimenzionálními parametry a projektem EvoApproxLib výzkumné skupiny EHW@FIT, který obsahuje stovky implementací různých aritmetických obvodů.
2. Navrhněte interaktivní webové rozhraní, které bude umožňovat vizualizovat parametry těchto obvodů a usnadňovat jejich výběr s ohledem na kritéria zadaná uživatelem (chybové metriky, plocha, příkon, zpoždění, apod.).
3. Zpracujte studii na výše uvedená témata.
4. Navržený systém implementujte v jazyce JavaScript. Parametry obvodů budou specifikovány ve formátu JSON a budou uloženy na webovém serveru. Při návrhu se zaměřte se na co nejefektivnější implementaci z pohledu nároků na HW zdroje klienta a dále jednoduchost ovládání.
5. Vyhodnoťte a diskutujte parametry implementovaného řešení.

Literatura:

- MRÁZEK Vojtěch, HRBÁČEK Radek, VAŠÍČEK Zdeněk a SEKANINA Lukáš. EvoApprox8b: Library of Approximate Adders and Multipliers for Circuit Design and Benchmarking of Approximation Methods. In: Proc. of the 2017 Design, Automation & Test in Europe Conference & Exhibition (DATE). Lausanne: European Design and Automation Association, 2017, s. 258-261.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 a 2 zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Vašíček Zdeněk, doc. Ing., Ph.D.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

prof. Ing. Lukáš Sekanina, Ph.D.
vedoucí ústavu

Abstrakt

Zámerom tejto práce je návrh a implementácia interaktívneho webového rozhrania, ktoré umožní užívateľom vybrať z knižnice tisícok implementácií obvodov aproximovaných násobičiek a sčítačiek pre návrh obvodov a porovnávaní, vyvíjaných výskumnou skupinou Evolvable Hardware (EHW) z Fakulty Informačných Technológii na Vysokom Učení Technickom v Brne. V práci sa možno dočítať o webových technológiách použitých pre realizáciu tejto práce. Taktiež možno nájsť kapitolu venujúcu sa návrhu vhodného užívateľského rozhrania a efektívneho algoritmu na spracovanie a vizualizáciu niekoľko tisícok implementácií.

Abstract

The desire of this thesis is to design and implement an interactive web interface that will allow users to select from a library of thousands of implementations of approximate adders and multipliers for circuit design and benchmarking, developed by Evolvable Hardware Group (EHW) from the Faculty of Information Technology at the Brno University of Technology. We will find out about the web technologies used for this thesis. Also, we will find a chapter devoted to designing a suitable user interface and an efficient algorithm to process and visualize several thousand implementations.

Kľúčové slová

web, aplikácia, JavaScript, React, Redux, D3.js, ECMAScript

Keywords

web, application, JavaScript, React, Redux, D3.js, ECMAScript

Citácia

SLÁVIK, Denis. *Interaktivní webové rozhraní pro výběr vhodných architektur*. Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Ing. Zdeněk Vašíček, Ph.D.

Interaktivní webové rozhraní pro výběr vhodných architektur

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Doc. Ing. Zdeněk Vašíček, Ph.D. a uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Denis Slávik
13. mája 2018

Poďakovanie

Chcel by som sa touto cestou poďakovať svojmu vedúcemu práce, pánovi docentovi Vašíčkovi za odborné vedenie a cenné rady pri tvorbe tejto práce. Veľká vďaka patrí aj celej komunite vývojárov zo sveta JavaScript, ktorí svojou tvorbou dávajú nový rozmer dnešnému webu, ako takému. Taktiež by som rád poďakoval svojim blízkym za morálnu podporu pri písaní tejto práce.

Obsah

Kapitola 1

Úvod

Žijeme v dobe, kde sa na vývoj webových aplikácií a technológií dáva čoraz väčší dôraz dôsledkom internetového rozmachu. Deň čo deň, viac a viac užívateľov využíva služieb internetu pre zjednodušenie a zefektívnenie ich každodenného života. Práve preto vývoj a prístup k webovým technológiám naberá obdivuhodné tempo a pozornosť programátorskej komunity.

Veľkú rolu pri rozmachu technológií spôsobili aj nové zariadenia na trhu, ktoré veľmi rapídne získali svoju popularitu medzi užívateľmi, ako smartfóny a tablety. Tieto zariadenia dovoľujú užívateľom takmer neustály prístup k informáciám prostredníctvom internetu.

Príchodom nových zariadení vznikol tlak na vývojárov od verejnosti pre optimalizáciu a neustále vylepšovanie užívateľského rozhrania pre zlepšenie skúsenosti s ich webovými stránkami. Komfort užívateľa spočíva v tom, aby nezáležalo na tom, odkiaľ, na akom zariadení a za akých podmienok užívateľ pristupuje k ich produktu, výsledok nech je vždy dobrý dojem a interakcia užívateľa s ich aplikáciou.

Obsah tejto práce rozoberá návrh a implementáciu moderného interaktívneho webového rozhrania, ktoré umožní užívateľovi hladkú a jasnú interakciu pri výbere vhodných implementácií elektronických obvodov používaných v digitálnej elektrotechnike.

V druhej kapitole 2 sa možno dozvedieť viac informácií ohľadne knižnice EvoApproxLib, kde sa popisuje pôvod a význam tejto knižnice. V kapitole je spomenutý momentálny stav webovej vizualizácie tejto knižnice a následná predstava, ako by sa mohla táto implementácia vylepšiť a zefektívniť.

Tretia kapitola 3 zahŕňa poznatky o rôznych technológiách, ktoré sa s vývojom rozhrania spájajú. Tieto technológie sú odzrkadlenia aktuálnych trendov vo svete webového vývoja na scéne JavaScriptových aplikácií.

Kapitola štvrtá 4 priblíži návrh rozhrania s odôvodneniami pri jednotlivých častiach aplikácie. Jej obsah zhrnie návrh nielen užívateľského rozhrania zo strany klienta, ale taktiež návrh fungovania prostredia potrebného pre správny chod aplikácie.

Ďalšia kapitola 5 rozoberá implementačnú časť realizácie tejto práce. V nej sa môžete stretnúť s približným popisom implementácie jednotlivých komponent rozhrania. Kapitola obsahuje aj zoznam s popisom použitých knižníc a nástrojov nespomenutých v predchádzajúcich kapitolách.

Ako poslednú kapitolu 6 nájdeme zhodnotenie a záver tejto práce.

Kapitola 2

Knižnica EvoApproxLib

Ide o knižnicu aproximovaných sčítačiek a násobičiek pre návrh obvodov a porovnaní [8]. Je určená pre ľudí z priemyslu, ktorí sa zaujímajú o aproximované obvody a metódy ich návrhu. Knižnica slúži na zrýchlenie návrhu týchto obvodov a systémov. Knižnica vznikla na pôde Fakulty Informačných technológií na Vysokom Učení Technickom v Brne zásluhou niektorých členov výskumnej skupiny Evolvable Hardware ústavu počítačových systémov.

2.1 Pôvodná podoba vizualizácie

Pred započatím tejto práce existovala verzia práce znázornená na obrázku C.1. Táto forma avšak mala mnoho nedostatkov, ktoré požadovali návrh novej dynamickej verzie.

Nedostatky starej verzie:

- Zastaralý vzhľad
- Neinteraktívny spôsob filtrovania údajov
- Náročnosť orientácie medzi parametrami
- Neefektívna vizualizácia veľkého množstva dát
- Neresponzívny vzhľad (fungujúci len pri veľkom rozlíšení)

2.2 Cielová predstava vizualizácie

Ako je jasné z predchádzajúcej časti, pôvodná vizualizácia je nedostatočná a ťažko osloví záujemcov. Preto po konzultácii s členmi výskumnej skupiny sme prišli k niekoľko nápadom ako vyvinúť vhodnú verziu vizualizácie knižnice.

Požadované zmeny na aplikácii:

- Modernejší, aktualizovaný vzhľad
- Možnosť lepšej interakcie užívateľa s aplikáciou
- Jednoduchá orientácia medzi prvkami aplikácie
- Lepšia vizualizácia dát pomocou grafov a kompaktného zoznamu výsledkov

- Možnosť filtrácie obvodov podľa viacerých parametrov
- Použitie aktuálnych webových technológií
- Možnosť dynamického načítania externých dátových súborov

Ďalšie zmeny a vylepšenia budú popísané v nasledujúcich kapitolách, po detailoch.

Kapitola 3

Webové technológie

V tejto kapitole si rozoberieme najzaujímavejšie technologické prvky použité pri realizácii tejto práce a odôvodnenie ich použitia. Pri technologickom rozmachu ktoré webové služby zažívajú, neustále sa vyvíjajú a pribúdajú nové technologické vymoženosti hotové na použitie vývojárom.

Úvodom sa zameriame na rolu skriptovacieho jazyku JavaScript, v ktorom je väčšina knižníc a doplnkov použitých v tejto práci, naprogramovaných alebo rozširujúcich jeho funkcionalitu. Tieto knižnice nám umožňujú čo najefektívnejšie tvoriť moderné webové aplikácie pre široký rozsah zariadení.

3.1 JavaScript

Skriptovací jazyk JavaScript posledné desaťročie pozoruhodným spôsobom získava čoraz väčšiu a väčšiu pozornosť komunity ľudí vyvíjajúcich moderné webové riešenia. Spolu s HTML a CSS tvoria hlavné prostriedky pre tvorbu webových aplikácií. V dávnejších dobách bol tento jazyk pokladaný skôr za prostriedok pre oživenie webových stránok prostredníctvom animácií prvkov a podobných techník. Avšak pozornosťou komunity sa presadzuje ako plnohodnotný skriptovací jazyk na tvorbu nielen front-endovej stránky vývoja, ale taktiež aj back-endovej, kde začína konkurovať jazykom ako PHP alebo Java.

Narastajúcou pozornosťou komunity a požiadavkami na vývojárov od užívateľov aplikácií sa jazyk ustavične posúva dobrým smerom, čo vedie k vzniku nových technológií, nástrojov a prístupov k tvorbe webových aplikácií. Týmto sa vylepšujú aj webové prehliadače, ktoré vylepšujú rýchlosť a vymoženosti aplikácií bežiacich v ich rozhraní. Za spomenutie však stojí aj to, že JavaScript neslúži len na strane prehliadačov a technológia Node.js [1] umožňuje vývoj back-end aplikácií, ktoré môžu bežať v pozadí na serveroch. Navyše sú tieto aplikácie výkonné a rýchle vďaka *V8 JavaScript engine* [2] od spoločnosti Google, ktorý používa aj ich webový prehľadávač Chrome.

3.1.1 ECMAScript 2015 (ES6)

Pri neustálom vývoji a pozornosti venovanej jazyku JavaScript bola potreba jazyk štandardizovať za účelom zjednotiť jazyk [10], aby pre vývojárov napr. prehliadačov bol určený istý záchytný bod. ECMAScript 2015 (taktiež známy ako ES6) je šiestou vydanou špecifikáciou jazyka JavaScript od asociácie European Computed Manufactures Association (ECMA).

[1] Technológia Node.js https://nodejs.org/en/
[2] Technológia V8 https://developers.google.com/v8/

Ako z názvu vyplýva, táto verzia štandardu bola vydaná v roku 2015 v mesiaci Jún. Ide o štandard ECMA-262 [7] publikovaný organizáciou Ecma International - organizácia vytvárajúca technologické štandardy.

Táto verzia priniesla do sveta JavaScriptu mnoho vylepšení a konštrukcií [3] a priviedla tak celkom novú vlnu modernizácie tohto jazyka.

Okrem verzie ES6 sú už k dispozícii vydania ES7 alebo ES8, avšak podpora špecifikovaných štandardov v rozsahu prehľadávačov ešte nie je úplná [3], preto sa pre produkčné verzie webových aplikácií používa šieste vydanie.

3.1.2 Babel

Pri plnení tohoto zadania sa snažíme zaručiť čo najrozsiahlejšiu kompatibilitu aplikácie na rôznych prehľadávačoch. Aj keď vyššie popísaný ECMAScript 6 (3.1.1) je plne podporovaný na väčšine globálne používaných prehľadávačoch, žiaľ niektoré z nich [3] (Internet Explorer a Edge od spoločnosti Microsoft), ešte neposkytujú plnú podporu rozšírení zo špecifikácie ES6.

Našťastie tento problém rieši kompilátor Babel [4], ktorý kompiluje kód novej generácie JavaScriptu do nižších implementácií ako je verzia ES5. Verzia ES5 má takmer plnú podporu pri všetkých globálnych prehľadávačoch aj v nižších verziách, takže kód prevedený naň rozbehnú bez problémov.

```
1  // ES 6
2  [1, 2, 3].map(n => n ** 2);
3
4  // ES 5
5  [1, 2, 3].map(function (n) {
6    return Math.pow(n, 2);
7  });
```

Výpis 3.1: Ukážka prevodu kódu medzi ES6 a ES5

3.2 Single Page Application (SPA)

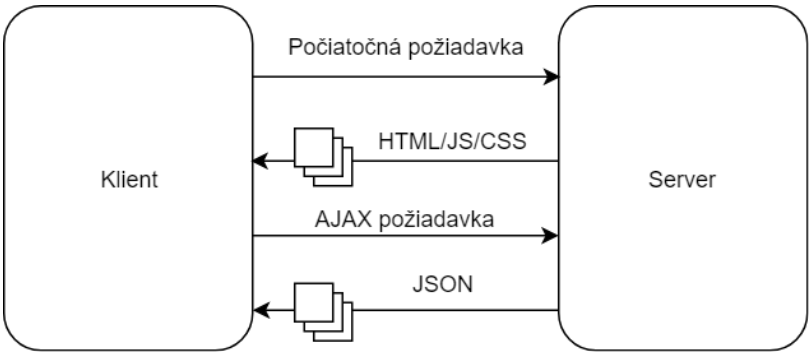
Ako sme boli pri prehľadávaní webových aplikácií zvyknutí, každá väčšia časť webovej aplikácie mala svoju vlastnú stránku, ktorá bola dostupná cez svoju unikátnu adresu. Problémom týchto stránok bolo, že pri každom prekliknutí na inú stránku sa znova a znova posielala požiadavka na server a vždy sa muselo čakať na dodanie veľkého objemu dát a stránka sa musela znova vykresliť s dodanými dátami.

S príchodom technológie akou je *Asynchronous JavaScript and XML* (AJAX)[5] sa naskytla možnosť pristupovať k návrhu webových aplikácií z iného smeru. Z tohto smeru vznikol neobvyklý model webových aplikácií, pod názvom Single Page Application alebo One Page Application. Tento prístup sa zameral na riešenie problému s prenosom veľkých dát pri prechode medzi jednotlivými stránkami aplikácie. Ďalšou z hlavných výhod tohto prístupu je aj zrýchlenie interakcie návštevníkov stránok s webovou aplikáciou.

Pri využití tejto metódy, užívateľovi je dopriaty skôr pocit, ako keby používal desktopovú aplikáciu a nie naopak webovú, ako tomu bol zvyknutý. Obsah, ktorý je nevyhnutný pre použitie stránky (JavaScript, HTML a CSS súbory) sa získa pri prvom načítaní stránky

[3] Pluginy vydania ECMAScript 6 https://babeljs.io/docs/plugins/#es2015
[4] Kompilátor Babel https://babeljs.io/
[5] Single Page Application https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX

a zdroje, ktoré sú potrebné podľa interakcie užívateľa so stránkou, sa načítajú dynamicky počas využívania, pomocou už spomínaného AJAXu, ktorý na pozadí odošle asynchrónnu požiadavku serveru pre požadovaný obsah. Tento proces sa udeje bez toho aby sa celá stránka znova musela načítať a prekreslí sa len časť stránky, s ktorou pracuje užívateľ. Názornú ukážku fungovania komunikácie vidieť na obrázku 3.1.



Obr. 3.1: Ukážka fungovania komunikácie SPA so serverom

3.3 Front-end knižnice

Vo svete frontendového webového vývoja medzi knižnicami a frameworkami funguje aktívna konkurencieschopnosť. Schopnosti vývojárov sa v dnešných dňoch popisujú podľa toho, s akými frameworkami a knižnicami pracuje pri svojej práci a do akej hĺbky im rozumie a ovláda ich. Každým rokom pribúdajú rôzne projekty s rôznymi vymoženosťami.

V špičkových radách súťaživosti medzi projektmi nájsť názvy ako Angular [6], React alebo napríklad Vue.js [7]. Ide o moderné technológie využívané na tvorbu moderný SPA webových riešení. Viaceré štúdie [9][5] dokazujú, že práve knižnica React (3.4) je jednou z najobľúbenejších a najžiadanejších knižníc na trhu práce. Tieto tituly si za posledné roky vyslúžila neustálou údržbou a vývojom nových rozšírení, ktoré vývojárom prinášajú možnosť lepšieho a kvalitnejšieho vývoja webových aplikácií.

3.4 React

Pri realizácii tejto práce sme potrebovali technológie, ktoré nám zaručia rýchly vývoj webovej aplikácie, ktorá bude efektívne pracovať a vizualizovať aj veľké dáta. Potrebovali sme nástroj, ktorým bude možné postaviť aplikáciu, ktorá bude užívateľsky čo najprístupnejšia a bude podporovaná, na čo najväčšej škále zariadení. Preto sme sa rozhodli túto aplikáciu postaviť na jednej z najpopulárnejších knižníc dnešnej doby, ktorou je React.

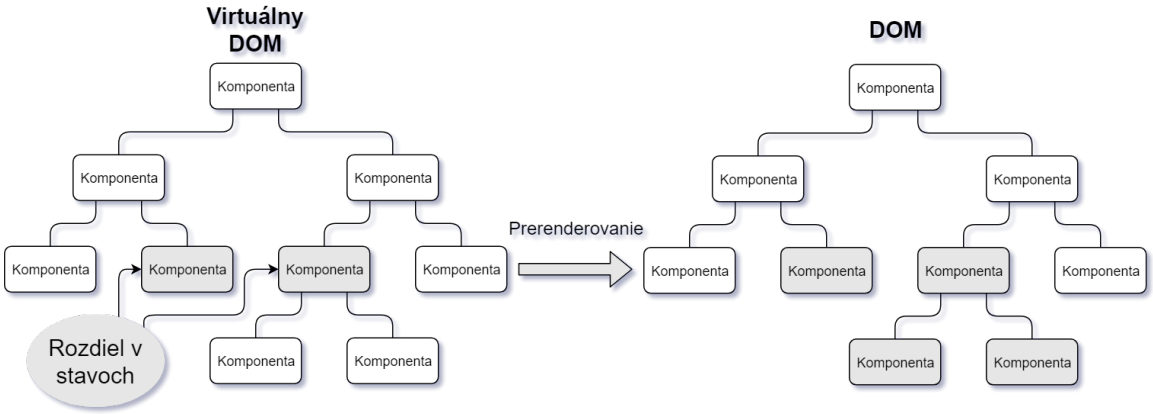
Knižnica React [8], ktorú si často ľudia mýlia s frameworkom, je pôvodom od vývojárov z gigantu, akým je Facebook. Umožňuje okamžitú tvorbu jednostránkových aplikácií (Single Page Applications) a flexibilnú tvorbu užívateľského rozhrania. Kľúčovými výhodami

[6] AngularJS https://angularjs.org/
[7] Vue.js https://vuejs.org/
[8] React - JavaScript knižnica https://reactjs.org/

knižnice React sú virtuálny DOM (Document Object Model) [9], životný cyklus komponent a plná správa vykresľovania spomínaných komponent.

3.4.1 Virtuálny DOM

Použitie Virtuálneho DOMu nám spôsobí, že sa v pamäti RAM vytvorí kópia DOMu prehľadávača, takzvaného BOM (*Browser Object Model*), a pri transformácii stavu jedných z komponent, z ktorého sa model skladá, dôjde k výpočtu rozdielu medzi vyrenderovaným modelom a novým modelom. Následne knižnica React sa postará o to, aby sa znova prerenderovali len časti, ktorých sa zmeny stavu týkajú. Týmto sa zabráni tomu, aby sa celá stránka musela znova a znova prekresľovať do prehliadača, a taktiež sa zabezpečí účinné fungovanie aplikácie. Fungovanie vykreslovania komponent pri zmene stavov je možné vidieť znázornené na obrázku 3.2.



Obr. 3.2: Ukážka fungovania renderovania komponent knižnice React pri zmene stavu

3.4.2 JSX

Nasledujúcou súčasťou knižnice React je využitie rozšírenia JSX (*JavaScript Syntax Extension*), ktorá nám rozširuje syntax jazyka HTML o mnoho funkcionalít a je akýmsi prepojením medzi JavaScript a HTML. Týmto pádom nám dovoľuje písanie akéhosi HTML kódu priamo do renderovacej funkcie komponent knižnice a vytvárať tak elementy aplikácie. Kód písaný v JSX je nakoniec transformovaný do JavaScriptu pomocou knižnice Babel.

```
1   const formatName = (user) => `${user.firstName} ${user.lastName}`;
2
3   const user = {
4     firstName: 'Dolan',
5     lastName: 'Duck'
6   };
7
8   const element = (
9     <h1>Welcome, {formatName(user)}!</h1>
10  );
11
12  ReactDOM.render(
```

[9] Document Object Model https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

```
13      element,
14      document.getElementById('root')
15    );
```

Výpis 3.2: Ukážka kódu JSX s podporou ES6

3.4.3 Životný cyklus komponenty

Komponenty, ako hlavné prvky systému React, majú metódy životného cyklu, ktoré slúžia na zachytenie jednotlivých etáp životnosti týchto komponent.

Metódy jednotlivých fáz[12]:

1. Mounting (fáza prvého vykreslenia komponenty):
   - *constructor()* – konštruktor komponenty, zavolá sa ako prvá metóda.
   - *componentWillMount()* – metóda, ktorá sa volá pred prvým vykreslením.
   - *render()* – metóda ktorá by mala vracať kód JSX na vykreslenie.
   - *componentDidMount()* – metóda volaná po prvom úspešnom vykreslení.
2. Updating (fáza po prvom vykreslení komponenty):
   - *componentWillReceiveProps()* – metóda sa volá pri každom prijatí alebo zmene parametra.
   - *shouldComponentUpdate()* – metóda, ktorá je volaná pri každom prekreslení, a dáva vývojárovi možnosť podľa stavu aplikácie samému rozhodnúť či je prekreslenie nutné alebo nie.
   - *componentWillUpdate()* – metóda volaná pred potvrdeným budúcim prekreslením.
   - *render()*
   - *componentDidUpdate()* – metóda volaná po prekreslení komponenty.
3. Unmounting (fáza pri odstraňovaní komponenty z modelu):
   - *componentWillUnmount()*
4. Error handling (fáza pri zachytení chyby vykresľovania komponenty):
   - *componentDidCatch()*

Je vhodné ešte uviesť, že pri našej práci používame knižnicu React vo verzii 16.2.0, pretože sa od verzie 16.3 menili celoplošne vyššie uvedené metódy, kde niektoré z nich sa už nepoužívajú a boli uvedené nové náhradné metódy [10].

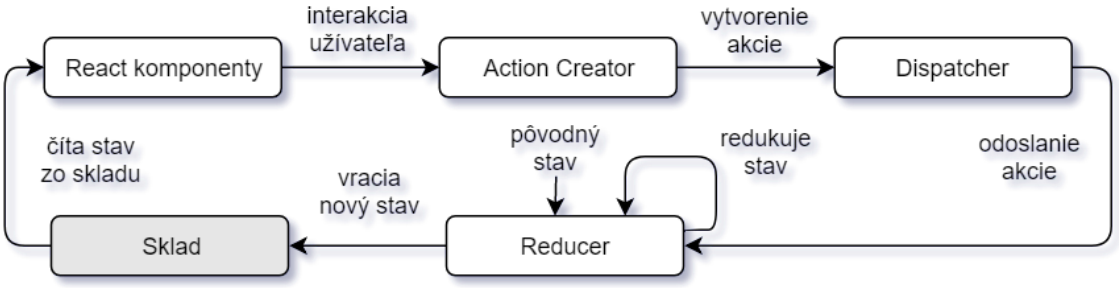
[10] React v16.3.0: New lifecycles and context API https://reactjs.org/blog/2018/03/29/react-v-16-3.html

3.5 Redux

Pri vývoji webových aplikácií sa nám stáva, že časom ako sa aplikácia rozrastá a naberá na komplikovanosti, naša aplikácia musí spravovať čoraz väčší vnútorný stav. Tento stav môže obsahovať napríklad odpovede servera alebo lokálne užívateľské dáta, s ktorými užívateľ aktívne pracuje a nie sú uložené v žiadnej databáze [2].

Knižnica Redux nám umožňuje vytvoriť stavový kontajner pre naše JavaScriptové aplikácie, ktorý bude globálne prístupný a čitateľný z ktorejkoľvek komponenty našej aplikácie. Tento stav sa správa deterministicky a predikovateľne vďaka funkciám knižnice.

Fungovanie reduxového stavu nám určuje jednosmerný tok dát, kde je vždy jasné, kde, v akom stave a aké dáta sú uchovávané 3.3.



Obr. 3.3: Jednosmerný tok dát za použitia knižnice Redux

Funkčnosť knižnice Redux sa dá popísať v troch kľúčových princípoch:

- „Jediný zdroj pravdy“ – stav celej aplikácie je uložený v jedinom stromovitom objekte, ktorý sa nazýva *Store*.
- Stav aplikácie je iba čitateľný – jediný spôsob ako zmeniť stav aplikácie je vyslať takzvanú *Action*, ktorá ako objekt popisuje o akú zmenu žiada užívateľ. Týmto spôsobom je vždy jasné, čo a akým spôsobom zmenilo stav aplikácie, a umožňuje nám to jednoduché hľadanie chýb.
- Zmeny sa vykonávajú prostredníctvom funkcií bez vedľajších účinkov – takéto funkcie nazývame *Reducer*. Reduceri nám udávajú, akým spôsobom sa zmení stav aplikácie.

Akcia

Akcia (*Action*) v knižnici Redux predstavuje objekt s dátami, ktoré sú predané z komponenty aplikácie a ich cieľom je modifikovať alebo byť uložené v sklade Reduxu (3.5).

Objekt akcie okrem samotných dát, musí obsahovať atribút *type*, ktorý popisuje typ akcie, aby sa vedel identifikovať spôsob, akým Reduceri (3.5) s týmito dátami budú modifikovať náš Sklad.

Existujú rôzne spôsoby akým sa tok dát dá zrealizovať. Jedným zo spôsobov, ktorý využijeme aj pri našom projekte je, namiesto priameho volania akcií, volajú funkcie zvané ako *action creator*. Tieto funkcie môžu byť volané ako klasické funkcie, takže napríklad pri interakcii užívateľom alebo pri tom ako sa nám naša komponenta načítala.

```
1  // Action creator for showing/hiding loader flag
2  export const setLoader = (name, show = 'true') => {
```

```
    return async dispatch => {
      if (show) {
        dispatch({
          type: FLAGS_LOADER_START,
          loader: name
        });
      } else {
        dispatch({
          type: FLAGS_LOADER_STOP,
          loader: name
        });
      }
    };
  };
```

Výpis 3.3: Ukážka action creatora s podmienenými dispatchermi

Reducer

Reducer v knižnice Redux sa zastáva funkcie zachytávania vytvorených akcií a podľa ich typu menia jednotlivé časti stavu aplikácie. Reducer pri zachytení akcie identifikuje o aký typ operácie ide a prostredníctvom modifikácie predošlého stavu aplikácie, vráti nový stav aplikácie s pridanou hodnotou.

Názov Reducer pochádza z funkcie jazyku JavaScript *Array.prototype.reduce*, ktorá prechádza položkami poľa a aplikuje na nich predom zvolenú funkcionalitu.

```
  // Reducer with initial state
  export default (state = { loaders: [] }, action) => {
    switch (action.type) {
      // Action to start loader
      case FLAGS_LOADER_START:
        return {
          ...state,
          loaders: [...state.loaders, action.loader]
        };
      // Action to stop loader
      case FLAGS_LOADER_STOP:
        return {
          ...state,
          loaders: without(state.loaders, action.loader)
        };
      default:
        return state;
    }
  };
```

Výpis 3.4: Ukážka reducera

Sklad

Sklad (angl. *Store*) nám poslúži ako úložisko dát, potrebných pre správny beh aplikácie, reprezentovaný ako objekt.

Použitie s knižnicou React

Síce knižnica Redux je nezávislá od všetkých iných knižníc, ale je ju možné použiť s hocijakým iným frameworkom, ako napríklad v našom prípade je knižnica React. Toto všetko nám umožňuje knižnica react-redux [11], ktorá nám poskytuje komponenty a funkcie na prepojenie týchto dvoch knižníc.

3.6 D3.js

D3 (*Data-Driven Documents*) alebo aj D3.js [12] je JavaScript knižnica pre interaktívnu vizualizáciu dát prostredníctvom webových prvkov, ako SVG [13], Canvas [14] a HTML [15].

Tempom, akým sú dáta v dnešnej dobe generované, nastáva problém tento veľký obsah dát prezentovať prijateľnou formou pre užívateľov. Vizuálna prezentácia dát je jedna z najefektívnejších spôsobov na poskytovanie informácií užívateľov a práve knižnica D3js poskytuje skvelé prostriedky na vytvorenie takýchto vizualizácií. Knižnica nám dovolí implementovať dynamické, intuitívne a rýchle vizualizácie dát.

Použitie s knižnicou React

Pri integrácii knižnice D3.js a React nastáva situácia, kde sa musíme rozhodnúť pre istý prístup súhry, pretože obe knižnice modifikujú DOM. V našom prípade, by sme mali prenechať celkovú správu stavu DOM knižnici React [13].

Je mnoho spôsobom ako k tomuto problému pristupovať, no my sme si zvolili spôsob, kde budeme knižnicu D3 používať na matematické výpočty a aktualizovanie prvkov, ktoré sme prenechali knižnici React na vytvorenie.

[11] Knižnica react-redux https://github.com/reactjs/react-redux
[12] Knižnica D3.js https://d3js.org/
[13] SVG - Scalable Vector Graphics https://developer.mozilla.org/en-US/docs/Web/SVG
[14] HTML <canvas> element https://developer.mozilla.org/en-US/docs/Web/HTML/Element/canvas
[15] HTML (HyperText Markup Language) https://developer.mozilla.org/en-US/docs/Glossary/HTML

Kapitola 4

Návrh

V tejto kapitole je možné nájsť popis celej etapy návrhu realizácie projektu. Pred návrhom aplikácie bolo potrebné analyzovať požiadavky na vylepšenie webovej vizualizácie knižnice EvoApproxLib so samotnými tvorcami tejto knižnice, ktoré sme uviedli v zozname 2.2.

V kapitole sa zameriame na analýzu zadania aplikácie 4.1, návrh toku a formátu dát, ktoré budú nevyhnutné pre beh aplikácie 4.2 a nakoniec sa zameriame na návrh jednotlivých komponent pre splnenie požiadaviek na vizualizáciu dát z knižnice EvoApproxLib s odôvodnením jednotlivých návrhov.

4.1 Zadanie aplikácie

Zo zadania a predošlých kapitol je jasné, že pôjde o webovú aplikáciu pre odborníkov z praxe, ktorí majú záujem o implementácie aproximovaných obvodov. Všetky nižšie uvedené návrhy vznikli po konzultáciách s členmi výskumnej skupiny, ktorá knižnicu EvoApproxLib dodáva.

Dáta, ktoré sú dodané z knižnice EvoApproxLib, by mali byť poskytnuté užívateľovi aplikácie čo najpríjemnejším spôsobom na výber. Užívateľ by mal mať na výber z rôznych typov implementácií podľa typu obvodu, ktoré budú uložené a načítavané z externého zdroja výskumnej skupiny Evolvable Hardware.

Po výbere vyhovujúcich skupín zariadení zo všetkých dostupných datasetov, by mal byť užívateľ schopný efektívne vyfiltrovať čo najvhodnejšie implementácie pre ich požiadavky. Filtrovanie by malo byť vizualizované pomocou histogramov, grafov a výsledky filtrovania by mali byť zobrazené v tabuľke výsledkov na konci užívateľského rozhrania. Parametre filtrovania budú, taktiež ako datasety, definované v externom súbore.

Tieto prvky určené na filtráciu veľkého množstva dát by mali byť intuitívne jednoduché na ovládanie užívateľom a mali by poskytovať širokú škálu filtračných funkcií, ako sú napríklad možný výber rozsahu dát pri histogramoch, alebo približovanie a posúvanie sa v grafoch zobrazujúcich výsledky vyhľadávania.

4.2 Formát a tok dát

V tejto sekcii si preberieme formát externých dát, ktorý pri návrhu bol vybraný. Taktiež sa dozvieme akým spôsobom bude s dátami počas behu aplikácie narábané.

4.2.1 Formát dát

Pred začatím realizácie projektu bolo nevyhnutným krokom určiť formát dát, ktoré budú aplikácii dodávané z externých zdrojov. Vzhľadom na to, že ide o JavaScriptovú webovú aplikáciu, jednou z najvhodnejších a najpoužívanejších formátov dát je formát *JavaScript Object Notation (JSON)*.

Ide o textový štandard, ktorý bol implementovaný za cieľom ukladania a výmeny štrukturovaných dát. Je alternatívnym riešením pre formát XML [1]. Jej hlavnými výhodami je jednoduchá parsovateľnosť, čitateľnosť a zapisovateľnosť. Syntax JSON je založený na objektoch jazyka JavaScript, ale v textovom formáte [6].

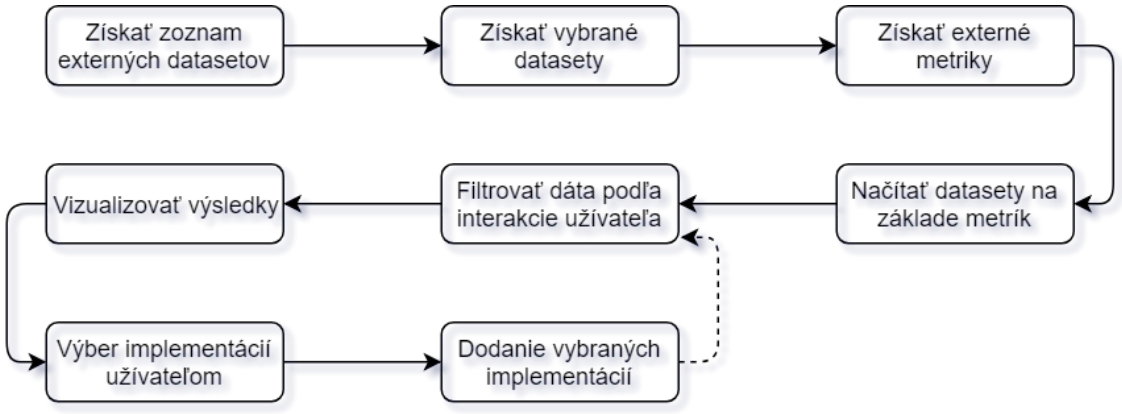
Výstupné dáta, ako vidieť na obrázku 4.1, budú vybrané implementácie užívateľom a budú dodávané z externej knižnice EvoApproxLib ako archív Zip [2], obsahujúci textové súbory s vybranými implementáciami.

4.2.2 Tok dát

Pri návrhu aplikácie bolo taktiež dôležitou časťou dobre zvážiť, akým spôsobom budú externé dáta spracované a ako s nimi budeme formovať pri užívateľskej interakcii. Po dohode s členmi výskumnej skupiny sme sa zhodli že okrem zoznamu dát z knižnice EvoApproxLib budú u nich hosťované aj údaje s metrikami, ktoré sa budú používať ako typy položiek pre filtračné prvky, alebo ako kolónky zobrazených údajov v tabuľke s výsledkami.

Ako vidieť na obrázku 4.1, proces sa začne získaním zoznamu všetkých datasetov z externých zdrojov. Následne podľa údajov v získanom zozname, sa po jednom načítajú externé datasety. Zároveň sa s externými datasetmi sa stiahne aj súbor s definovanými metrikami, ktoré budú použité napríklad pre filtračné komponenty.

Po získaní dát sa užívateľovi aplikácia načíta a môže začať s interakciou. Budú mu poskytnuté vhodné komponenty na filtráciu datasetov. Po každej zmene sa užívateľovi budú vizualizovať vhodné implementácie podľa kritérií. Tieto implementácie budú vyberateľné užívateľom a bude im poskytnutá možnosť si tieto implementácie stiahnuť do svojich zariadení.



Obr. 4.1: Tok dát počas behu aplikácie

[1] XML - Extensible Markup Language https://developer.mozilla.org/en-US/docs/XML_introduction
[2] Zip - formát súboru https://en.wikipedia.org/wiki/Zip_(file_format)

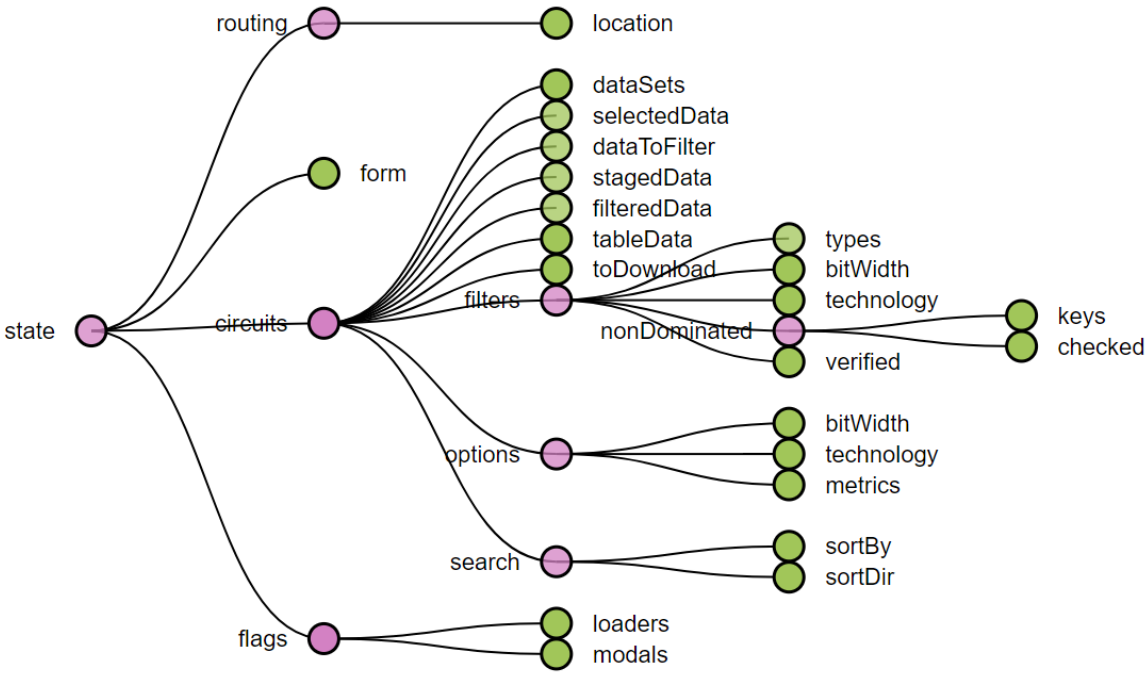
4.2.3 Návrh stavu aplikácie

Neodmysliteľnou súčasťou návrhu infraštruktúry našej aplikácie bolo uvedomiť si, aké lokálne dáta bude užívateľ aplikácie potrebovať počas behu aplikácie. Ako sme pri predošlej kapitole určili, o správu a udržiavanie lokálneho stavu aplikácie budeme používať knižnicu redux (3.5).

Počas návrhu stavu bolo nutné si ozrejmiť aké dáta bude užívateľ počas používania aplikácie potrebovať.

Ako vidieť na finálnom návrhu stavu 4.2, náš stav bude zložený z podstromov:

- *routing*: stav „smerovača“ spravovaný a vytvorený knižnicou react-router-redux 5.6.2
- *form*: podstrom, ktorý bude obsahovať hodnoty a stavy formulárových vstupov aplikácie spravovanej a vytvorenej knižnicou redux-form (5.6.1).
- *flags*: vlastný podstrom, ktorý bude obsahovať jednoduché príznaky používané užívateľským rozhraním, ako sú príznaky načítavania externých údajov alebo toho, že práve na pozadí prebieha filtrovanie dát.
- *circuits*: vlastný podstrom a zároveň jeden z hlavných nosičov dôležitých užívateľských dát. Bude uschovávať všetky datasety, filtrované údaje alebo napríklad aj údaj, ktoré implementácie boli zvolené užívateľom na stiahnutie. Taktiež bude obsahovať dôležité údaje pre filtračné prvky užívateľského rozhrania.



Obr. 4.2: Návrh počiatočného stavu aplikácie

4.3 Uživateľské rozhranie

Neodmysliteľnou súčasťou návrhu aplikácie bolo dôkladné rozmyslenie ako užívateľovi podať možnosť pracovať s dátami a ako ich vizualizovať. V tomto prípade sa teda pozrieme na koncept návrhu užívateľského rozhrania a teoretického návrhu jednotlivých hlavných komponent. Popis návrhu jednotlivých komponent bude v poradí, ako tomu pri návrhu skutočne bolo.

Semantic UI

Existujú skvelé riešenia ako urýchliť prototypovanie a návrh užívateľských rozhraní. Pri návrhu rozhraní webových aplikácií sú veľmi nápomocné CSS frameworky, ktoré nám umožnia vďaka predpripraveným komponentám svižnejšiu prípravu náhľadu webových stránok.

V dnešnom rozmachu technológií, je na výber mnoho CSS frameworkov. Pre naše účely sme si vybrali populárny framework s názvom Semantic UI. Okrem moderného dizajnu širokej škály komponentov nám Semantic UI pomôže aj už existujúcou verziou pre React, ktorá nám týmto pádom ešte väčšmi uľahčí návrh rozhrania. Naviac od niektorých známych CSS frameworkov, tento framework je možné použiť bez použitia jQuery, čo je hlavným princípom pri použití Reactu [1].

Ďalšími výhodami Semantic UI frameworku je možnosť vytvárať plne responzívne moderné webové rozhrania. Samozrejme pre tých, ktorý chcú tento framework použiť pri klasickom vývoji webových aplikácií, je v k dispozícii „nereactovská“ verzia [3].

Medzi hlavné prvky aplikácie budeme zaraďovať tieto komponenty:

- Komponenta pre výber datasetov
- Komponenty pre výber bitovej dĺžky a cieľovej technológie
- Komponenty pre výber nedominantných a formálne verifikovaných implementácií
- Komponenta s histogrammi pre výber určitého rozsahu implementácií na základe parametra
- Komponenta s vizualizáciou vyfiltrovaných dát prostredníctvom bodových grafov
- Komponenta tabuľky pre zobrazenie vyfiltrovaných dát podľa externých metrík s možnosťou označovať implementácie
- Komponenta pre možné stiahnutie vybratých implementácií

4.3.1 Výber datasetov

Ako prvým prvkom, na ktorý sme sa pri návrhu zamerali, bola komponenta, ktorá užívateľovi dovolila výber medzi načítanými datasetmi podľa kategórie obvodov. Táto komponenta by mala byť umiestnená niekde na popredí medzi prvými filtračnými prvkami.

Komponentu sme navrhli ako vstupný prvok s vizážou rozbaľovacieho menu (angl. *drop-down menu*), ktoré bude obsahovať zoznam všetkých dostupných datasetov zaradených do kategórií podľa typu implementácie s možnosťou označenia alebo odznačenia istých setov. Označenie a odznačenie navrhneme ako prvky zaškrtávacích políčok (angl. *checkbox*).

[3] Semantic UI https://semantic-ui.com/

Každému zvolenému datasetu by mala byť pridelená unikátna farba, ktorá ešte žiadnej inej pridelená nebola, aby sa dali implemetácie z istých datasetov neskôr rozoznať napríklad pri komponente bodových grafov.

Taktiež bolo predom určené, aby aktívne označené datasety pochádzali z tej istej kategórie, keďže nemá žiaden význam mať v jednej filtrácii implementácie dvoch alebo viacerých rôznych typov.

Finálnu podobu realizácie návrhu popísanej kompoenty možno vidieť na obrázku C.2.

4.3.2 Výber bitovej dĺžky a cieľovej technológie

Tieto prvky budú slúžiť pre výber istého druhu implementácií. Budú implementované ako rozbaľovacie menu, ale s fixným výberom len jednej z možností. Dostupné možnosti týchto menu budú nutné vždy generovať pri výbere nových datasetov, z dôvodu, že každá kombinácia datasetov môže mať iné dostupné riešenia v rôznych bitových šírkach alebo v rôznych cieľových technológiách.

4.3.3 Výber nedominantných riešení a formálne verifikovaných implementácií

Nasledujúce prvky budú slúžiť takisto filtračným účelom. Prvok formálne verifikovaných riešení by mal zabezpečiť, že sa užívateľovi vyfiltrujú len tie implementácie, ktoré majú medzi svojimi dátami určitý indikátor (angl. *flag*) nastavený ako pravdivý. Tento prvok bude navrhnutý ako jednoduché zaškrtávacie políčko.

Čo sa týka prvku pre výber nedominantných riešení to bude o niečo komplikovanejšie. Táto komponenta okrem zaškrtávacieho políčka bude potrebovať niečo ako rozbaľovacie menu, s možnosťou označenia minimálne troch parametrov, podľa ktorých sa vyberú iba nedominantné implementácie z datasetov.

4.3.4 Histogramy pre výber určitého rozsahu implementácií

Prvý z prvkov pre vizualizáciu dát s filtračnými účelmi, bude navrhnutý ako histogram. Nepôjde o jeden histogram, ale o obmedzený zoznam histogramov, kde každý bude môcť filtrovať podľa vybratého parametru.

Histogramy by mali umožňovať okrem vizualizácie dát podľa parametru aj výber určitého rozsahu. Výber rozsahu sme navrhli ako prvok pre výber rozsahu. Okrem funkcionality výberu rozsahu, by malo byť možné histogramy medzi sebou presúvať a tak určovať ich prioritu filtrácie dát. To znamená, že sa dáta budú filtrovať postupne po vybratých aktívnych rozsahoch jeden po druhom. Napríklad výber rozsahu pri prvom histograme ovplyvní dáta vo všetkých ostatných histogramoch.

Histogramy by mali podľa návrhu taktiež disponovať funkcionalitou pre približovanie (angl. *zoom*) a posúvanie (angl. *panning*) vo zvolených rozsahoch keďže sa vo zvolenou rozsahu môže nachádzať ešte priveľa implementácií a užívateľ by potreboval upresniť daný rozsah ešte viac do hĺbky a tak vybrať užší rozsah.

Okrem tejto zásadnej funkcionality bude možné histogramy dynamicky pridávať a odstraňovať medzi sebou, taktiež zrušiť vybraný rozsah dát alebo zrušiť priblíženie a posúvanie pomocou tlačidiel.

Výber typu parametra podľa ktorého sa budú dáta zobrazovať navrhneme ako rozbaľovacie menu s možnosťou zvoliť parameter.

Finálnu podobu realizácie návrhu popísanej kompoenty možno vidieť na obrázku C.3.

4.3.5 Vizualizácia pomocou bodových grafov

Po vyfiltrovaní dát predchádzajúcimi prvkami by sme chceli užívateľovi vizualizovať vybrané implementácie. Jedným z najvhodnejších spôsobov ako podľa istých kombinácií parametrov vizualizovať neurčité kvantum implementácie boli bodové grafy.

Tieto bodové grafy sme navrhli, aby zobrazovali vyfiltrované implementácie z rôznych datasetov. Každý bod by mal byť zafarbený podľa pridelenej farby datasetu.

Medzi hlavnými funkcionalitami možno zaradiť:

- Dynamické pridávanie a odstraňovanie grafov
- Výber parametrov osí X a Y
- Výber typu zobrazenia osi podľa lineárnej, logaritmickej alebo exponenciálnej stupnice
- Interaktívne približovanie a posúvanie sa v grafoch
- Možnosť výberu implementácie na stiahnutie priamo z grafu
- Vykreslenie pareto fronty [4]

Finálnu podobu realizácie návrhu popísanej kompoenty možno vidieť na obrázku C.4.

4.3.6 Tabuľka pre zobrazenie dát

Okrem vizualizácie výsledkov filtrácie prostredníctvom bodových grafov sme sa rozhodli aj pre klasickú komponentu v tvare tabuľky, kde budú zobrazené jednotlivé implementácie a všetky dostupné informácie o nej.

Tabuľka by mala mať tieto funkcionality:

- Možnosť výberu implementácií na stiahnutie užívateľom
- Možnosť radiť výsledky podľa zvoleného parametru z hlavičky

Finálnu podobu realizácie návrhu popísanej kompoenty možno vidieť na obrázku C.5.

4.3.7 Komponenta na stiahnutie implementácií

Výstupom aplikácie, ako sme už spomínali v predchádzajúcich kapitolách, by mali byť okrem vizualizácie vybraných implementácií aj možnosť si tieto implementácie ako archív stiahnuť.

Pre tento účel sme navrhli komponentu, ktorá bude neustále viditeľná pre užívateľa. Pôjde o tlačidlo, ktoré bude zobrazovať počet označených implementácií na stiahnutie. Ak žiadna implementácia označená nebude, užívateľovi sa bude zobrazovať možnosť stiahnuť si všetky implementácie zo zvolených datasetov.

[4] Pareto fronta https://en.wikipedia.org/wiki/Pareto_efficiency#Pareto_frontier

Kapitola 5

Implementácia

Pri tejto kapitole sa zameriame na tie najdôležitejšie aspekty pri implementácii riešenia našej práce. Popíšeme si utilitu na vytvárania React aplikácií zvanú *create-react-app* a vysvetlíme si jej výhody a obsah. Ďalej sa zameriame na mechanizmus získavania externých údajov a proces filtrovania údajov. Zameriame sa aj na implementáciu hlavných prvkov webovej aplikácie, komplikácie pri ich tvorbe a akým spôsobom sme tieto komplikácie vyriešili.

5.1 create-react-app

Utilita s názvom create-react-app [1] bola vytvorená rovnakými vývojármi, ktorý vyvíjajú knižnicu React. Utilita slúži, ako je z názvu jasné, na vytvorenie React aplikácie.

Jedna z hlavných problémov, na ktorú sa užívatelia neustále sťažovali ohľadne Reactu, bola náročnosť začiatkov zakladania projektov postavených na tejto knižnici a potreba poznať mnoho ďalších knižníc, ktoré boli potrebné pre vytvorenie funkčnej verzie aplikácie [11]. Táto utilita užívateľovi neuveriteľným spôsobom uľahčí založenie základu svoju projektu bez nijakej znalosti knižníc a systému, ktoré pracujú na pozadí.

Utilita vytvorí funkčný projekt, ktorý bude obsahovať časti, ktoré budú pre nás užitočné:

- React (3.4), JSX (3.4.2), Babel (3.1.2), ES6 (3.1.1) podpora.
- Automaticky prefixované CSS pomocou Autoprefixer (5.6.5).
- Prednastavený skript, určený pre vybudovanie produkčnej verzie. Skript je ovládaný cez webpack (5.6.3).
- „Live development server“ – lokálne bežiaci server, určený pre vývoj aplikácie prostredníctvom webpack-dev-server (5.6.4).

Utilita nám ponúka samozrejme ďalšie vymoženosti, avšak pre náš účel nie sú za potreba použitia, preto sme ich zo zoznamu vyššie vynechali.

[1] Utilita create-react-app https://github.com/facebook/create-react-app

5.2 Získanie a spracovanie externých dát

Jednej z prvých vecí, ktoré bolo nevyhnutné implementovať do našej aplikácie, bola funkčnosť získať dáta z externých zdrojov a ich spracovanie do našej aplikácie.

Ako sme pri návrhu našej aplikácie spomenuli, všetky externé dáta budú lokálne skladované po získaní v našom Redux sklade (3.5). Pri načítaní našej hlavnej komponenty, ktorá bude obaľovať všetky ostatné, zavoláme akciu (3.5), ktorá bude mať za úlohu získať externý zoznam dostupných datasetov.

5.2.1 Fetch API

Štandardným spôsobom získavania externých údajov v JavaScript aplikáciách sa pôvodne používal koncept XMLHttpRequest [2], ktorý avšak prichádzal z veľmi chaotickým kódom. Následne sa začala používať metóda *ajax* z knižnice jQuery [3], ktorá prinášala svojou funkcionalitou lepšie riešenie problému, ale stále sa jednalo o neštandardnú funkciu z knižnice tretej strany.

Nová doba moderného JavaScriptu nám prináša ako súčasť jazyka takzvané Fetch API. Fetch API nám prináša rýchlu a prehľadnú možnosť vytvárať požiadavky po sieti [4]. Toto vylepšenie jazyka JavaScript disponuje mnohými vylepšeniami a prichádza z vymoženosťami ako používanie Promises [4] alebo rozdelenou logikou medzi sieťovou požiadavkou a odpoveďou [5].

```
1  fetch('https://example.com/data.json', { method: 'GET', headers: customHeaders })
2    .then(function (response) {
3      // Transform response to json
4      return response.json();
5    })
6    .then(function (data) {
7      // Read data as json
8      console.log(data);
9    })
10   .catch(function (err) {
11     // Catch error if somethings goes wrong
12     console.log("Something went wrong!", err);
13   });
```

Výpis 5.1: Ukážka použitia Fetch API

Využitie pri implementácii

V našom prípade budeme používať Fetch API pre získavanie externých dát – zoznamu datasetov, jednotlivých datasetov, metrík pre filtrovanie.

5.2.2 Lodash

Lodash [6] je moderná JavaScript knižnica ponúkajúca desiatky optimalizovaných riešení pre rôzne úlohy v dnešných JavaScript aplikáciách. Umožňuje zjednodušenie práce s poliami, číslami, objektami, reťazcami a mnoho ďalším.

[2] XMLHttpRequest (XHR) https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest
[3] jQuery API Dokumentácia – ajax() http://api.jquery.com/jquery.ajax
[4] JavaScript Promises https://developers.google.com/web/fundamentals/primers/promises
[5] The Fetch API https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
[6] Lodash knižnica https://lodash.com/

Využitie pri implementácii

V našej aplikácii nám budú nápomocné jej funkcie pri riešení problémov ako filtrovanie údajov podľa istých parametrov určených užívateľom, zoraďovanie výsledkov filtrácie pre užívateľa podľa vybraného atribútu alebo pri optimalizácii často volaných funkcií.

5.3 Implementácia filtračných komponent

Filtračné komponenty našej aplikácie budú slúžiť na vizualizáciu dát s možnosťou výberu jednotlivých parametrov s cieľom zúžiť výber ponúkaných výsledných implementácií.

5.3.1 Výber datasetov

Komponentu pre výber datasetov sme implementovali ako komponentu kompatibilnú a pripravenú na použitie s API knižnice redux-form (5.6.1). Z hľadiska užívateľského rozhrania sme použili na komponentu kombináciu predpripravených komponent frameworku Semantic UI (4.3), v popredí komponentu rozbaľovacieho menu (*Dropdown*) a zaškrtávacieho políčka (*Checkbox*).

Možnosti rozbaľovacieho menu datasetov na výber, komponenta číta z redux skladu, kde sa zapíšu pri načítaní z externých zdrojov. Komponenta taktiež pri interakcii užívateľa vyvoláva redux akcie, ktorým predáva nové zvolené datasety. Po zmene datasetov sa prepočítajú všetky ostatné filtračné údaje a parametre, ktoré závisia na zvolených dátach.

Implementácia komponenty obsahuje tieto vymoženosti:

- Výber maximálne 6 datasetov zároveň
- Výber datasetov iba z rovnakých kategórií
- Označenie/odznačenie všetkých datasetov z istej kategórie
- Dynamické prideľovanie farieb aktívnym datasetom
- Zobrazenie chybového stavu, pri ktorom nie je žiaden dataset aktívny alebo nie je žiaden k dispozícii

5.3.2 Výber bitovej dĺžky a cieľovej technológie

Tieto komponenty sme implementovali ako rozbaľovacie menu (*Dropdown*) zo Semantic UI, pričom sme zamedzili výber z možností na jednu položku.

Obe komponenty čítajú svoju aktuálnu hodnotu a možnosti výberu zo skladu Redux. Možnosti na výber v oboch prípadoch sa vypočítavajú pri zmene vybratých datasetov. Prepočítavajú sa preto, pretože každý dataset môže obsahovať implementácie z rôznymi parametrami.

Zápis nových hodnôt je zabezpečený prostredníctvom vyvolania action creatora, ktorému sa prepošle typ filtra a nová hodnota. Pri zmene ktorejkoľvek z hodnôt sa zároveň prepočítajú možnosti pre ostatné filtračné komponenty.

5.3.3 Výber nedominantných riešení

Komponenta na výber formálne verifikovaných implementácií je filtračnou komponentou implementovanej ako zaškrtávacie políčko (*Checkbox*). Pri zaznamenaní zmeny hodnoty na za-

škrtnutý stav (*typ Boolean, hodnota true*) sa vyfiltrujú všetky implementácie z aktívnych datasetov, ktorých atribút *verified* je hodnoty rovnajúcej sa reťazcu *all*.

5.3.4 Histogramy s výberom rozsahov

Prvým komplexnejším filtračným prvkom, ktorý sa pri implementácii naskytol, bola komponenta, ktorá nám podľa vybraného parametru implementácií, vizualizovala vybraté dáta pomocou histogramu a pomocou sliderov nám umožnila výber určitého úseku z ponúknutých dát.

Komponenta je implementovaná spôsobom, že nám umožní dynamicky pridávať, meniť a mazať jednotlivé histofiltre. Hodnota, ktorú táto komponenta vytvára bude prezentovať pole objektov, ktoré budú identifikovať typ zvoleného filtra a aktuálne zvolené hraničné hodnoty z rozsahov.

Implementácia histogramov

Pri implementácii časti vizualizácie dát histogramom sme použili JavaScript knižnicu D3.js 3.6. Prvou užitočnou funkciou ktorú sme použili, bola funkcia *histogram* [7], ktorá nám z dodaných dát dokázala vygenerovať dáta potrebné pre vytvorenie histogramov.

Následné získané pole dát, ktoré predstavovalo dátový popis jednotlivých pruhov histogramu sme prostredníctvom technológií, ktoré nám ponúka webový štandard SVG (*Scalable Vector Graphics*) a jej prvky, vykreslili do našej komponenty jednotlivé pruhy nášho histogramu.

Aby bolo jednoznačné, pre matematické výpočty sme použili knižnicu D3.js, ale samotné grafické spracovanie sme zvládli prostredníctvom React komponenty, kde sme v JSX formáte použili prvky SVG.

Implementácia výberu rozsahu

Komponenta slúžiaca na výber rozsahov na sliderech nie je žiadnym štandardne implementovaným prvkov, preto sme si ho museli od základu navrhnúť a vytvoriť sami.

Pri implementácii nám bola nápomocná funkcia *brushX* [8] z knižnice D3js. Táto funkcia nám dovoľuje vytvorenie jedno dimenzionálneho „štetca“ (angl. *brush*) na X-ovej osi. To znamená, že nám vytvorí prvok, ktorým budeme môcť označovať isté úseky na danom rozsahu.

Funkcia *brushX* nám bude zachytávať každú zmenu vybraného rozsahu. Okrem editácie hraničných hodnôt, nám dovoľuje aj presun celého rozsahu naraz.

Implementácia radenia histofiltrov

V návrhu sme sa zmienili o tom, že by mala existovať možnosť jednotlivé histofiltre medzi sebou prioritne radiť. Tento cieľ sme dosiahli tým, že každý jeden filter okrem dát ako typ filtru a hodnoty vybraného rozsahu, bude obsahovať aj niečo ako indexné číslo. Indexné číslo alebo ho môžeme nazvať aj prioritným číslom, nám bude určovať poradie aktuálneho filtru.

[7] D3.js - histograms https://github.com/d3/d3-array#histograms
[8] D3.js - brushing https://github.com/d3/d3-brush#d3-brush

Pre zaistenie funkcionality, ktorá nám dovolí filtre interaktívnym spôsobom medzi sebou premiestňovať, nám poskytla knižnica react-sortable-hoc [9]. Táto knižnica nám poskytuje komponenty, ktorými možno obaliť naše kompenenty filtrov, a spraví z nich animovaný triediteľný zoznam. Komponenta zachytáva jednotlivé fázy pri premiestňovaní filtrov a podľa počiatočného a finálneho umiestnenia, určí nové indexné čísla medzi filtrami a dáta vo filtroch sa prepočítajú navzájom podľa priority.

Približovanie a posúvanie

Knižnica D3.js ponúka riešenie taktiež na zachytenie užívateľských udalostí ako sú kliknutia myšou, pri mobilných zariadeniach rôzne gestá prstami a podobne. Nášmu návrhu, ktorý má zabezpečiť možnosť užívateľovi si približovať a posúvať sa vo vybranom rozsahu, dá vyhovieť prostredníctvom funkcie *zoom* [10], ktorá disponuje skvelou funkcionalitou pre splnenie nášho cieľa.

5.3.5 Filtrácia dát

Spôsob, akým sa deje filtrácia dát sa dá popísať tak, že hlavná komponenta odoberá každý stav obsahujúci hodnoty jednotlivých filtračných komponent popísaných vyššie. Ihneď ako komponenta prijme nový údaj, teda informáciu, že niektorý z hodnôt sa zmenil, zavolá action creator, ktorý následne prečíta všetky filtračné hodnoty vybrané užívateľom z aktuálneho stavu našej aplikácie a následne prevedie filtráciu všetkých implementácií z aktívnych datasetov, a vyšle reduceru akciu, ktorá popisuje zmenu nových výsledných implementácií.

Všetky komponenty, ktoré vizualizujú výsledky, čítajú práve tento stav, ktorý sa pri každej interakcii užívateľa s filtračnými prvkami mení na najaktuálnejšie výsledky. Týmto sa zabezpečuje jednoznačný tok dát a každá komponenta má k dispozícii rovnaké dáta.

5.4 Implementácia vizualizačných komponent

Po vyfiltrovaní vybraných implementácií užívateľov aplikácie, bolo treba zaručiť čo najefektívnejšiu vizualizáciu výsledkov. Keďže implementácie obsahujú mnoho atribút, po návrhu sme sa rozhodli implementovať pre vizualizáciu dát dynamické prvky ako bodové grafy s mnohými vymoženosťami a taktiež finálnu tabuľku výsledných implementácií.

5.4.1 Grafy

Prvom komponentou, ktorá bola navrhnutá ako jasná a intuitívna forma prezentácie výsledných implementácií je komponenta s dynamickými bodovými grafmi, ktoré nám poskytuje možnosť vytvorenia rôznych kombinácií grafov zobrazujúcich užívateľom žiadané dáta.

Implementácia grafov

Prostredníctvom dynamického pridávania, odoberania a editácie ponúkame užívateľom interaktívnu formu prezentácie dát. Toto je možné implementáciou komponenty, ktorá bude zobrazovať grafy, ktoré sú dátovo interpretované ako pole objektov nesúce o sebe informácie. Objekt, ktorý popisuje jednotlivé grafy, sa skladá z informácií o každej ose grafu. Každá

[9] Knižnica react-sortable-hoc http://clauderic.github.io/react-sortable-hoc/
[10] D3.js - zoom https://github.com/d3/d3-zoom#d3-zoom

informácia o ose, je prezentovaná znova ako objekt, ktorý nesie dáta ako typ osy a stupnica, podľa ktorej sa dáta na ose vykresľujú.

Pre implementáciu grafu použijeme znova knižnicu D3.js ako v predošlej sekcii. Nápomocné nám v prípade tvorby grafu budú funkcie z *d3-scale* [11] – pre implementáciu rôznych stupníc, alebo *d3-axis* – pre implementáciu vizualizácie ôs. Pre približovanie a posúvanie sa v grafe použijeme rovnaké funkcie ako sme použili pri implementácii histogramov (5.3.4).

Problémy pri implementácii

Jednou z hlavných problémov pri implementácii s ktorými sme sa stretli, bola práve vizualizácia dát s grafmi. Počiatočný prístup bol taký, že sme jednotlivé body grafu vykresľovali do DOM ako prvky SVG. Tento prístup fungoval v poriadku, avšak pri rozširovaní knižnice EvoApproxLib a zväčšovaní obsahu vizualizovaných dát, aplikácia prestávala byť rýchla a efektívna. Problémom bol veľký počet elementov (tisícky), ktorých sme sa snažili vykresliť do DOM.

Tento problém sa vyriešil spôsobom, pri ktorom sme sa rozhodli pre prechod z prvkov SVG na vykresľovanie do HTML5 elementu *<canvas>*, ktorý je prostredníctvom Canvas API [12] ovládateľný aj jazykom JavaScript.

Nasledujúci problém vznikol pri vyriešení predchádzajúceho problému. Síce sa aplikácia zrýchlila, ale vykresľovaním do elementu canvas sme stratili možnosť natívneho zachytávania kliknutí na vykresľované body. Zachytávanie kliknutí na jednotlivé body slúžil na označovanie/odznačovanie ľubovoľných implementácií na stiahnutie užívateľom.

Tento problém sa nám podarilo vyriešiť pomocou vytvorenia štruktúry dát *quadtree* [13] z knižnice D3.js. Následne sme na ploche grafu prostredníctvom funkcií *event* [14] zachytávali pozíciu zakliknutia a metódou *find()* dátovej štruktúry quadtree sme vyhľadávali zakliknuté body v grafe.

5.4.2 Tabuľka

Finálnym prvkom našej webovej aplikácie bola vizualizačná komponenta určená na zobrazenie veľkého množstva dát efektívne pre užívateľa. Týmto prvkom je tabuľka výsledkov implementácií.

Kľúčová funkcionalita implementácie

Tabuľku výsledkov našej aplikácie sme sa snažili aj napriek tomu, že má vizualizovať veľké množstvo dát, implementovať tým smerom, aby dáta boli pohodlne dostupné pre každého užívateľa na čo najširšej škále zariadení.

Problém predošlej verzie aplikácie bol ten, že z dôvodu veľkého počtu atribútov, ktoré jednotlivé implementácie mali, nebolo efektívne možné zobraziť všetky tieto dáta na jeden riadok bez toho aby sa aplikácia otvorila vo veľkom rozlíšení.

Riešenie tohoto problému nám ponúkla knižnica react-virtualized [15]. Táto knižnica ponúkla celú škálu komponentov pripravených na efektívnu vizualizáciu veľkého obsahu dát.

[11] D3.js - scales https://github.com/d3/d3-scale
[12] Canvas API https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API
[13] D3.js - quadtree https://github.com/d3/d3-quadtree
[14] D3.js - events https://github.com/d3/d3-selection#handling-events
[15] Knižnica react-virtualized https://github.com/bvaughn/react-virtualized

V našom prípade sme použili komponentu *Grid* [16], ktorá svojou výbornou mechanikou dovoľuje vykresľovať veľký počet dát. Tento systém funguje tak, že zaznamenáva pomocou JavaScriptu pozíciu, kde sa užívateľ nachádza vo vizualizovanom zozname a vykresľuje do DOM len elementy, ktoré sú momentálne užívateľom žiadané. V prípade, že užívateľ začne zoznamom prechádzať, začne sa zaznamenávať, na akej pozícii sa v zozname nachádza a podľa toho sa vypočítavajú elementy na vykresľovanie. Táto vymoženosť nám nepomôže len na dlhé zoznamy, ale taktiež na široké, ako tomu je v našom prípade.

Tabuľka je implementovaná ako dva Grid komponenty (takzvaný *MultiGrid*) [17]. Prvá, ktorá bude fixne umiestnená, nám bude zobrazovať identifikačné čísla jednotlivých výsledkov tabuľky, kde v druhom Gride budú zobrazené všetky ostatné atribúty výsledkov. Druhý Grid systém bude oproti prvému horizontálne rolovací. Týmto pádom budú výsledky čisto a prehľade prezentované užívateľom.

Ďalej v tabuľke sú implementované hlavičky, ktoré na kliknutie spôsobujú zoraďovanie jednotlivých výsledkov podľa zvoleného atribútu. Taktiež je súčasťou implementácie tabuľky vstupné pole, ktoré užívateľovi umožňuje podľa zadaných výrazov filtrovať výsledky v tabuľky.

5.5 Získanie balíku výsledkov

Finálnym krokom u užívateľa pri použití našej webovej aplikácie by mala byť možnosť stiahnuť si vybrané/všetky implementácie z dát, ktoré boli užívateľom vyfiltrované.

Implementácia sťahovanie výsledkov

Po tom, čo si užívateľ pri bodových grafoch alebo pri tabuľke výsledkov vybral implementácie na stiahnutie, mu bude k dispozícii v aplikácii tlačidlo, ktoré bude zobrazovať počet vybratých implementácií na stiahnutie.

Vybrané implementácie budeme ukladať v našom Redux sklade, vo formáte poľa obsahujúceho identifikačné čísla implementácií.

Po kliknutí na tlačidlo užívateľ svojou interakciou vyvolá akciu, ktorá prostredníctvom Fetch API (5.2.1) odošle službe hosťovanej výskumnou skupinou Evolvable Hardware Group, sieťovú žiadosť typu *POST* v tele so zoznamom požadovaných implementácií.

Služba následne ako odpoveď pošle súbor obsahujúci požadované dáta. Pre uloženie prijatých dát užívateľovi na jeho lokálny disk, použijeme knižnicu file-server [18].

```
1   import { saveAs } from 'file-saver';
2
3   const formData = new FormData();
4   formData.append('ids', JSON.stringify(circuitIdsToDownload));
5
6   fetch('https://example.com/download.php', { method: 'POST', body: formData })
7     .then(function (response) {
8       // Transform response to blob
9       return response.blob();
10    })
11    .then(function (data) {
12      // Save data as raw
```

[16] React-virtualized - Grid https://bvaughn.github.io/react-virtualized/#/components/Grid

[17] React-virtualized - MultiGrid https://bvaughn.github.io/react-virtualized/#/components/MultiGrid

[18] FileSaver.js https://github.com/eligrey/FileSaver.js

```
13        saveAs(data, 'circuits.zip');
14      })
15      .catch(function (err) {
16        // Catch error if somethings goes wrong
17        console.log("Something went wrong!", err);
18      });
```

Výpis 5.2: Ukážka získania a uloženia požadovaných dát

5.6 Ďalšie použité knižnice a nástroje

V tejto sekcii sú spomenuté ostatné knižnice a nástroje, ktoré boli použité pri návrhu a vývoji aplikácie a stoja za zmienku.

5.6.1 redux-form

Knižnica redux-form [19] nám poskytuje možnosť spolu s knižnicou react-redux (3.5), uschovávať stav našich formulárov v sklade knižnice redux (3.5). Naďalej nám poskytuje aj komponenty, ktoré nám dodávajú rozsiahle riešenia pre naše formulárové prvky.

5.6.2 react-router a react-router-redux

Tieto dve knižnice našej React webovej aplikácii pridajú funkcionalitu konfigurácie smerovania adries (angl. *routing*). To znamená, ak sa chceme navigovať cez túto React aplikáciu s viacerými stránkami, budeme potrebovať smerovač (angl. *router*) pre správu URL adries.

Knižnica react-router [20] sa nám teda postará o synchronizáciu užívateľského rozhrania a URL adresy, a knižnica react-router-redux [21] nám poskytne vždy aktuálny stav v našom redux sklade.

5.6.3 webpack

Nástroj s menou Webpack bol vytvorený za účelom tvorby balíkov moderných webových JavaScript aplikácií. Jej úlohou je vytvoriť balíky z AMD a CommonJS modulov pre škálu prehliadačov.

5.6.4 webpack-dev-server

Tento nástroj spolu s použitím nástroja Webpack (5.6.3) nám poskytuje server na vývoj našej aplikácie. Mal by sa používať výhradne na strane vývoja, napríklad na lokálnom stroji.

5.6.5 Autoprefixer

Plugin Autoprefixer [22] ako súčasť frameworku PostCSS [23] nám zabezpečuje automatické parsovanie CSS súborov a pridá potrebné zápisy s prefixami.

[19] Knižnica Redux Form https://redux-form.com
[20] Knižnica React-Router https://github.com/ReactTraining/react-router
[21] Knižnica react-router-redux https://github.com/reactjs/react-router-redux
[22] Plugin Autoprefixer https://github.com/postcss/autoprefixer
[23] Framework PostCSS https://github.com/postcss/postcss

```css
/* Vstupne CSS bez prefixu */
::placeholder {
  color: gray;
}

/* Vystup pluginu po~aplikovani a~pridani prefixov */
::-webkit-input-placeholder {
  color: gray;
}
:-ms-input-placeholder {
  color: gray;
}
::-ms-input-placeholder {
  color: gray;
}
::placeholder {
  color: gray;
}
```

Výpis 5.3: Ukážka fungovania pluginu Autoprefixer

Kapitola 6

Záver

V tejto práci sa bolo možné dozvedieť o štúdii a stave moderných webových aplikácií a technológií. Poskytla taktiež konkrétne informácie o technológiách akými sú JavaScript, ES6, Babel, React alebo aj Redux.

Bakalárska práca popísala tie najdôležitejšie časti návrhu a vývoja vizualizácie knižnice s názvom EvoApproxLib, ktorá obsahuje tisíce implementácií rôznych obvodov. Výsledkom je webová aplikácia s moderným interaktívnym prostredím, ktorá zvláda vizualizáciu parametrov obvodov a uľahčuje ich výber pre užívateľov, spôsobom rôznych intuitívnych filtračných prvkov.

Z pohľadu ďalšieho vývoja projektu existuje ešte mnoho možností ako sa dá webová aplikácia vylepšiť a rozšíriť o funkcionality, ktoré by mohli užívateľom ešte viac uľahčiť používanie apilkácie. Jednou z možností je aj implementácia webovej aplikácie ako PWA (Proressive Web App), ktorá by umožňovala off-line používanie bez dostupného internetového pripojenia. Ďalším možným rozšírením tejto aplikácie by bola implementácia funkcie zdieľania jednotlivých vyhľadávaní medzi užívateľmi.

Literatúra

[1] *Semantic UI React*. [Online; navštíveno 02.04.2018].
URL https://react.semantic-ui.com/

[2] *Motivation*. Marec 2018, [Online; navštíveno 02.04.2018].
URL https://redux.js.org/introduction/motivation

[3] Aranda, M.: *What's the difference between JavaScript and ECMAScript?* Október 2017, [Online; navštíveno 02.04.2018].
URL https://medium.freecodecamp.org/whats-the-difference-between-javascript-and-ecmascript-cba48c73a2b5

[4] Bangare, S.: *The Fetch API*. Marec 2018, [Online; navštíveno 02.04.2018].
URL https://medium.com/beginners-guide-to-mobile-web-development/the-fetch-api-2c962591f5c

[5] Elliott, E.: *Top JavaScript Libraries & Tech to Learn in 2018*. December 2017, [Online; navštíveno 02.04.2018].
URL https://medium.com/javascript-scene/top-javascript-libraries-tech-to-learn-in-2018-c38028e028e6

[6] Gabry, O. E.: *JSON In a Nutshell*. September 2016, [Online; navštíveno 02.04.2018].
URL
https://medium.com/omarelgabrys-blog/json-in-a-nutshell-7d638dfea7cc

[7] International, E.: *ECMAScript® 2015 Language Specification*. [Online; navštíveno 02.04.2018].
URL http://www.ecma-international.org/ecma-262/6.0/index.html

[8] Mrázek, V.; Hrbáček, R.; Vašíček, Z.; aj.: EvoApprox8b: Library of Approximate Adders and Multipliers for Circuit Design and Benchmarking of Approximation Methods. In *Proc. of the 2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, European Design and Automation Association, 2017, ISBN 978-3-9815370-9-3, s. 258–261.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=11262

[9] Park, J. S.: *The status of JavaScript libraries & frameworks: 2018 & beyond*. Marec 2018, [Online; navštíveno 02.04.2018].
URL https://medium.com/@alberto.park/the-status-of-javascript-libraries-frameworks-2018-beyond-3a5a7cae7513

[10] Prusty, N.: *Learning ECMAScript 6*. Packt Publishing Ltd., 2015, ISBN 978-1-78588-444-3.

[11] Richey, B.: *Learning React With Create-React-App*. November 2017, [Online; navštíveno 02.04.2018].
URL https://medium.com/in-the-weeds/learning-react-with-create-react-app-part-1-a12e1833fdc

[12] Szczeciński, B.: *Understanding React-Component life-cycle*. September 2017, [Online; navštíveno 02.04.2018].
URL https://medium.com/@baphemot/understanding-reactjs-component-life-cycle-823a640b3e8d

[13] Tiberghien, T.: *React + D3.js: Balancing Performance & Developer Experience*. Máj 2017, [Online; navštíveno 02.04.2018].
URL https://medium.com/@tibotiber/react-d3-js-balancing-performance-developer-experience-4da35f912484

Príloha A

Obsah priloženého CD

Obsah nosiča CD:

- technická správa práce vo formáte PDF
- Zdrojové súbory textu bakalárskej práce
- Zdrojové súbory bakalárskej práce s návodom na inštaláciu a vybudovanie produkčnej verzie

Príloha B

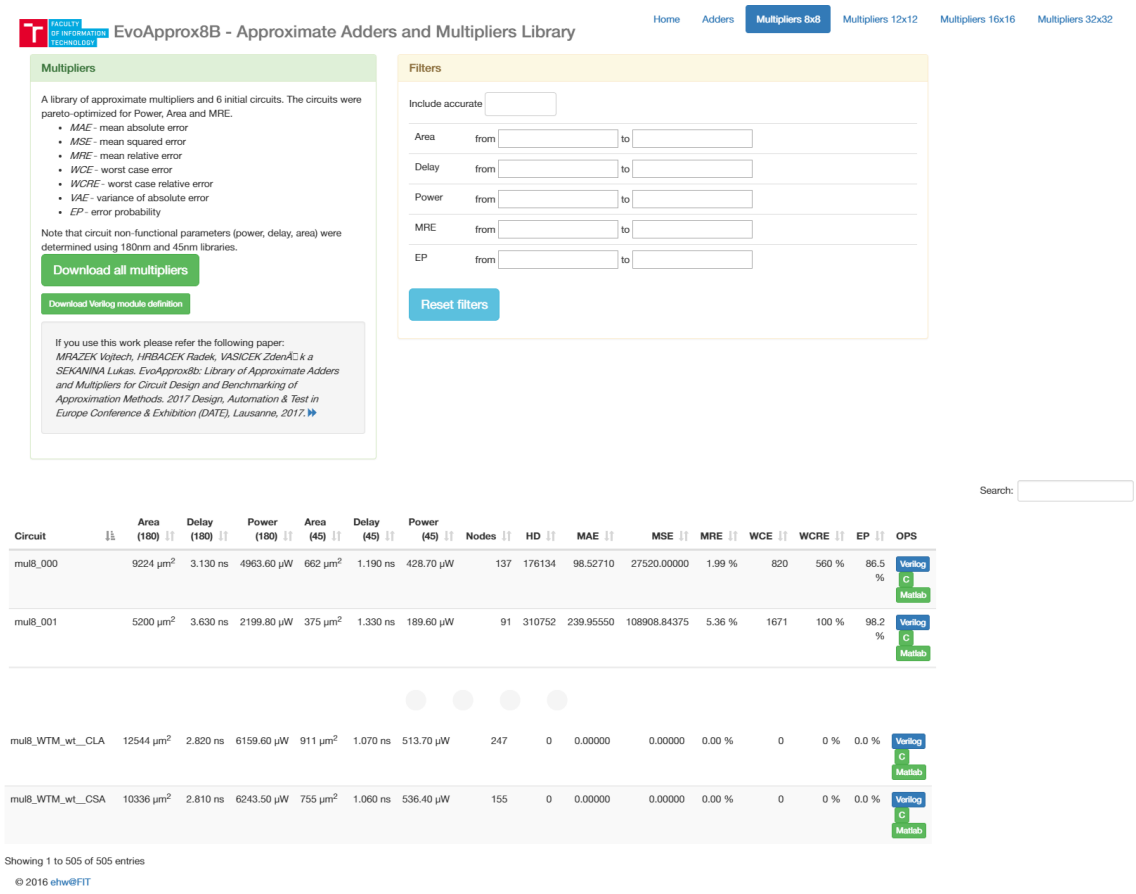
Obsah súboru package.json

```
{
  "name": "evo-approx-lib",
  "version": "1.0.0",
  "private": true,
  "homepage": "https://slavikdenis.github.io/evo-approx",
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "eject": "react-scripts eject",
    "predeploy": "yarn run build",
    "deploy": "gh-pages -d build"
  },
  "dependencies": {
    "babel-polyfill": "^6.26.0",
    "classnames": "^2.2.5",
    "d3": "^5.1.0",
    "dom-helpers": "^3.3.1",
    "file-saver": "^1.3.8",
    "gh-pages": "^1.1.0",
    "history": "^4.7.2",
    "lodash": "^4.17.10",
    "prop-types": "^15.6.1",
    "react": "^16.2.0",
    "react-dom": "^16.2.0",
    "react-loadable": "^5.4.0",
    "react-redux": "^5.0.7",
    "react-router-dom": "^4.2.2",
    "react-router-redux": "^5.0.0-alpha.9",
    "react-scripts": "1.1.4",
    "react-sortable-hoc": "^0.6.8",
    "react-virtualized": "^9.18.5",
    "redux": "^4.0.0",
    "redux-form": "^7.3.0",
    "redux-logger": "^3.0.6",
    "redux-thunk": "^2.2.0",
    "semantic-ui-css": "^2.2.14",
    "semantic-ui-react": "^0.80.0",
    "typeface-roboto": "^0.0.54"
  },
  "devDependencies": {
    "babel-eslint": "^8.2.3",
    "eslint": "^4.19.1",
```

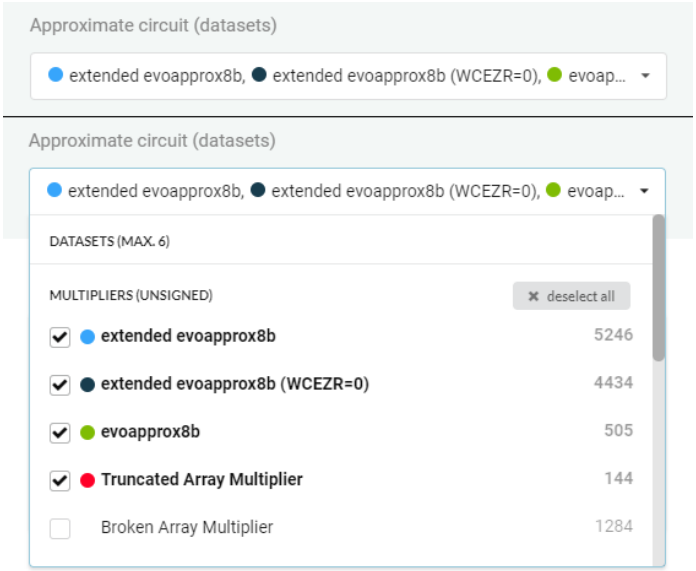
```
    "eslint-config-airbnb": "^16.1.0",
    "eslint-config-prettier": "^2.9.0",
    "eslint-config-react-app": "^2.1.0",
    "eslint-plugin-flowtype": "^2.46.3",
    "eslint-plugin-import": "^2.11.0",
    "eslint-plugin-jsx-a11y": "^6.0.3",
    "eslint-plugin-prettier": "^2.6.0",
    "eslint-plugin-react": "^7.7.0",
    "prettier": "^1.12.1"
  }
}
```

Príloha C

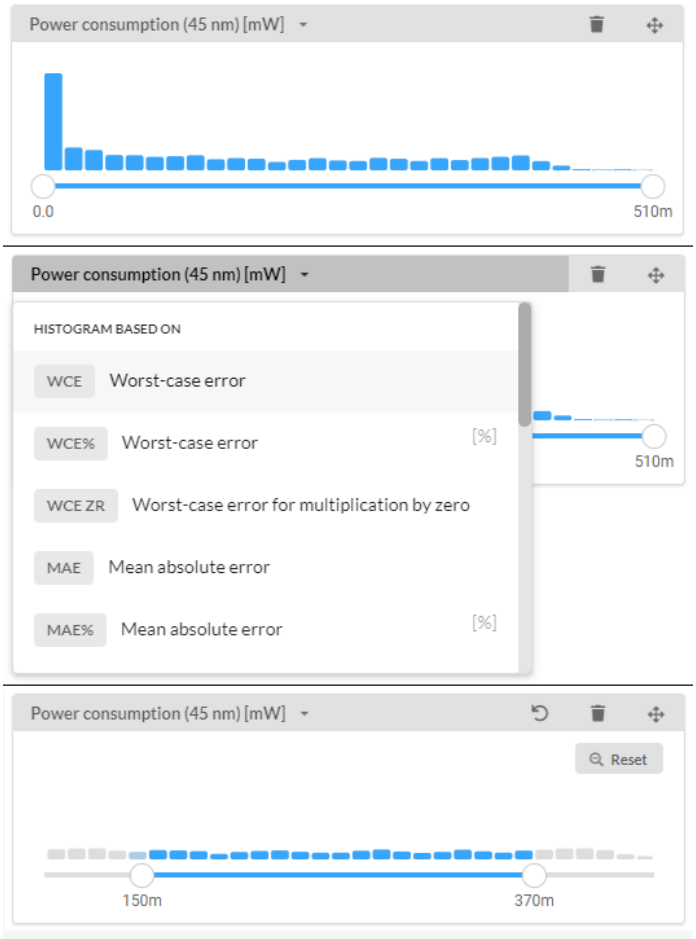
Obrázky



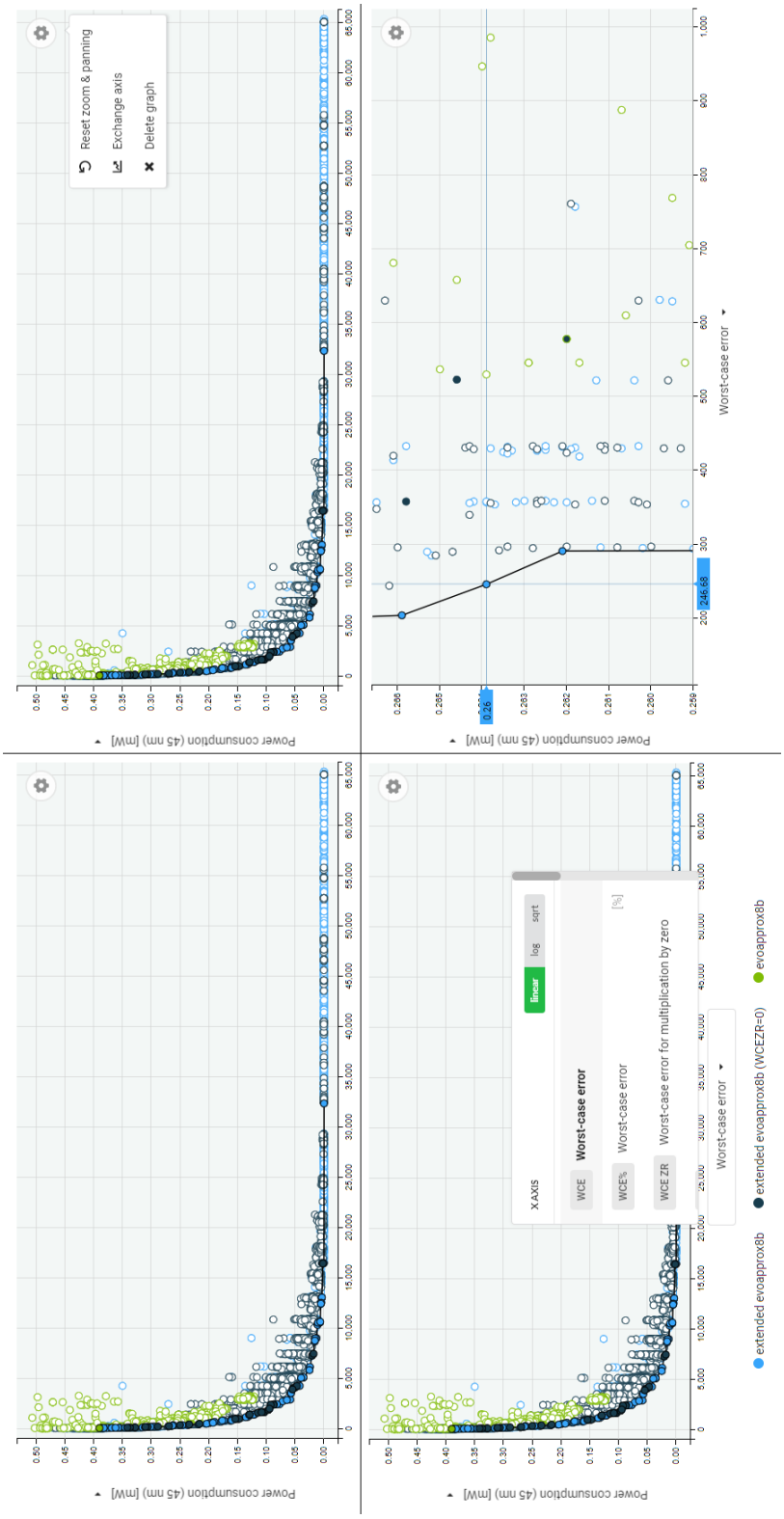
Obr. C.1: Celkový náhľad na starú verziu vizualizácie knižnice EvoApproxLib



Obr. C.2: Náhľad na rôzne stavy komponenty pre výber datasetov

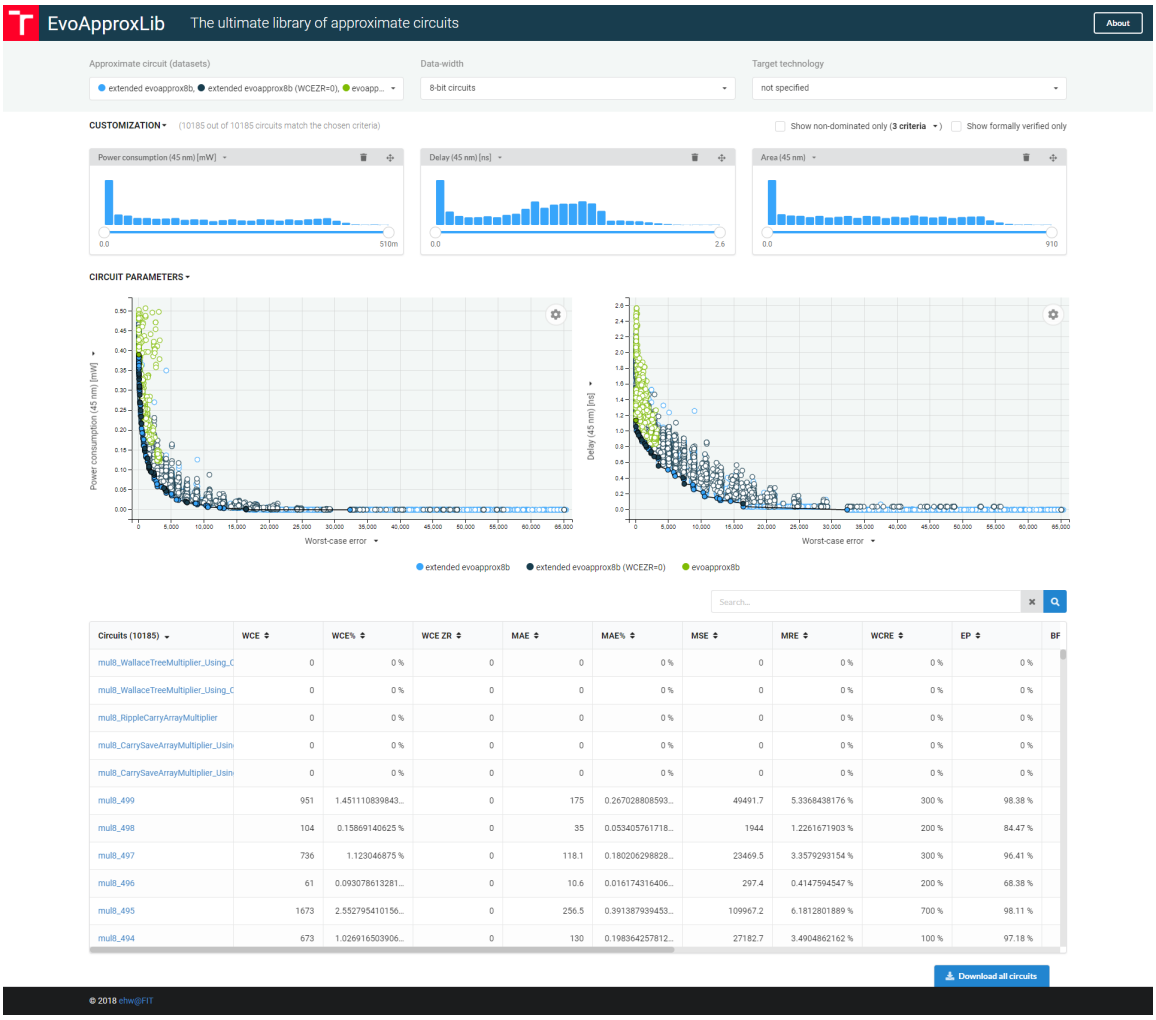


Obr. C.3: Náhľad na rôzne stavy komponenty histogramu



Obr. C.4: Náhľad na rôzne stavy komponenty grafov

Obr. C.5: Náhľad na rôzne stavy komponenty tabuľky s výsledkami



Obr. C.6: Celkový náhľad na novú verziu vizualizácie knižnice EvoApproxLib