F3

Fakulta elektrotechnická
Katedra kybernetiky

Bakalářská práce

Uživatelské rozhraní grafové databáze

Sergej Kurbanov
Otevřená informatika, Informatika a počítačové vědy
kurbaser@fel.cvut.cz

Květen 2018
Vedoucí práce: RNDr. Marko Genyk-Berezovskyj

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kurbanov** Jméno: **Sergej** Osobní číslo: **456113**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra kybernetiky**

Studijní program: **Otevřená informatika**

Studijní obor: **Informatika a počítačové vědy**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Uživatelské rozhraní grafové databáze

Název bakalářské práce anglicky:

Graph Database User Interface

Pokyny pro vypracování:

Navrhněte a implementujte webové uživatelské rozhraní pro databázi neorientovaných prostých grafů na serveru graphs.felk.cvut.cz. Rozhraní má umožnit uživateli transparentní přístup k aplikacím, které budou v databázi efektivně vyhledávat skupiny grafů pomocí co nejširšího výběru kriterií a zároveň umožní grafy zobrazovat a stahovat ve většině běžných formátů. Rozhraní rovněž poskytne přístup k interaktivnímu grafovému kalkulátoru, jehož pomocí bude uživatel určovat vlastnosti jím zadaných specifických grafů.
Databáze graphs.felk.cvut.cz postupně vzniká, soustřeďte se na spolupráci s API serverem, rozhraní konstruuujte tak, aby bylo co nejvíce flexibilní a podporovalo práci s postupně rostoucím objemem grafů i jejich vlastností v databázi.
Vyberte vhodné technologie podporující tvorbu tohoto druhu rozhraní, zdůvodněte jejich volbu a projekt doprovoďte dokumentací zaměřenou především na další možnosti rozšiřování funkcionality vaší aplikace.

Seznam doporučené literatury:

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein: Introduction to Algorithms, 3rd ed., MIT Press, 2009
[2] J. Matoušek, J. Nešetřil: Kapitoly z diskrétní matematiky, Karolinum, 2010
[3] R. Sedgewick: Algorithms in C Part 5: Graph Algorithms (3rd Edition), Addison-Wesley Professional, 2002
[4] J. Demel: Grafy a jejich aplikace, Praha, Academia, 2002

Jméno a pracoviště vedoucí(ho) bakalářské práce:

RNDr. Marko Genyk-Berezovskyj, katedra kybernetiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **12.01.2018** Termín odevzdání bakalářské práce: **25.05.2018**

Platnost zadání bakalářské práce: **30.09.2019**

RNDr. Marko Genyk-Berezovskyj	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Ing. Pavel Ripka, CSc.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.

Datum převzetí zadání Podpis studenta

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Poděkování / Prohlášení

Děkuji panu Berezovskému za vedení bakalářské práce. Děkuji kolegům Tomáši Rounovi a Herbertu Ullrichovi za spolupráci. Děkuji celému světu IT za pohon kupředu a za všechny úžasné vzniklé technologie.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 25.5.2018

Abstrakt / Abstract

Účelem této bakalářské práce je implementace přívětivého a zároveň mocného uživatelského rozhraní, které i běžným lidem umožní různé manipulace s existující databází grafů. Veliká data v dnešní době hrají velikou roli, a tak se práci s nimi dá vyhnout jen obtížně. Člověk ale standardně pojme jen omezené množství informací a pozornost nerozloží do více než hrstky směrů, proto při představení „problému" desítek milionů grafů, přichází řešení v podobě naší webové aplikace.
Konkrétními cíli jsou umožnit intuitivní plošné i detailní zobrazování grafů a jejich filtrování podle daných kritérií, dále jak jednotlivé, tak hromadné stahování pro pozdější práci s nimi na lokálním stroji a nakonec schopnost interakce s grafovým kalkulátorem počítajícím celou škálu vlastností grafů.
K optimálnímu splnění bylo použito nejmodernějších frontendových technologií. Jmenovitě HTML5, CSS3 a jeho preprocesor Sass, JS, framework React, knihovnu Redux a React Router. Vývoj nám významně usnadnil Node.js, Webpack a Babel.

Klíčová slova: Node.js, React, CSS, HTML, Webpack, frontend, Redux, JavaScript, React Router, Webpack

The purpose of this bachelor thesis is the implementation of a pleasant to use yet powerful user interface that allows users to operate with an existing graph database. Big data play a big role in today's day and age and so it's difficult to avoid working with them. A person can contain only a limited amount of information and one's attention can be divided only a handful of ways. When presented with a `problem` of tens of millions of graphs, we come with a solution of our web application.
Particular goals are to allow collective and individual display and filtering of graphs, ability to download graphs individually and in bulk for further work locally on a computer. Finally, to allow users to interact with a computer of a multitude of graph attributes.
During the development, these cutting-edge technologies were used: HTML5, CSS3 and it's preprocessor Sass, JS, React framework, Redux and React router libraries. Development was made a lot easier thanks to Node.JS, Webpack and Babel.

Keywords: Node.js, React, CSS, HTML, Webpack, frontend, Redux, JavaScript, React Router, Webpack
Title translation: Graph Database User Interface

Obsah /

1 Úvod . . . 1
2 Použité technologie . . . 3
2.1 HTML . . . 3
2.1.1 Elementy . . . 3
2.1.2 DOM . . . 4
2.1.3 HTML5 . . . 4
2.1.4 Alternativy . . . 4
2.2 CSS . . . 5
2.2.1 Pravidla . . . 5
2.2.2 Kompatibilita . . . 6
2.2.3 Knihovny . . . 6
2.2.4 Preprocesory . . . 6
2.2.5 Preprocesor Sass . . . 8
2.3 JavaScript . . . 8
2.3.1 Verze JavaScriptu . . . 8
2.3.2 Babel . . . 9
2.3.3 Zajímavosti . . . 9
2.4 Node.js . . . 9
2.4.1 npm . . . 10
2.5 React . . . 11
2.5.1 Virtual DOM . . . 11
2.5.2 JSX . . . 11
2.5.3 Komponenty . . . 12
2.5.4 Stav a event handlery . . . 12
2.5.5 Lifecycle metody . . . 13
2.6 Redux . . . 13
2.7 React Router . . . 15
2.8 Webpack . . . 15
2.9 Git . . . 15
3 Vlastní implementace . . . 16
3.1 Prostředí Node.js . . . 16
3.1.1 Struktura projektu . . . 17
3.2 Webpack . . . 17
3.2.1 Produkce . . . 18
3.2.2 Vývoj . . . 19
3.3 React . . . 19
3.4 Image.js . . . 20
3.4.1 CompletenessPage.js . . . 22
3.5 Redux . . . 24
3.6 React Router . . . 25
3.7 Css . . . 27
3.7.1 Fonty . . . 28
3.7.2 Scss . . . 28
4 Komunikace s backendem . . . 29
4.1 Fetch API . . . 29
4.1.1 Promise . . . 29
4.1.2 Použití v aplikaci . . . 29
5 Uživatelská příručka . . . 32
5.1 Webová aplikace . . . 32
5.1.1 Graphs . . . 32
5.1.2 Detail grafu . . . 33
5.1.3 About . . . 33
5.1.4 Complete collections . . . 34
5.1.5 Compute & insert . . . 34
5.2 Přidávání vlastností grafů . . . 35
5.2.1 config.js . . . 35
5.3 Přidávání formátů stahování . . . 35
5.3.1 Download.js . . . 36
6 Další rozšiřování a nedostatky . . . 37
6.1 Další rozšiřování . . . 37
6.2 Nedostatky . . . 37
7 Závěr . . . 38
Literatura . . . 39

Kapitola 1
Úvod

V době, kdy se digitální svět rapidně plní kvanty nových informací – ať už jako důsledek rozsáhlého sbírání dat o uživatelích všemožných zařízení, ale i lidech mimo technickou sféru, nebo kvůli nespočetným výzkumům majícím za účel lepší pochopení světa či nalezení vzorků chování ve společnosti a nebo prostě díky lidské kreativitě a široké přístupnosti internetu – může být lehké se v této bitové džungli ztratit. Proto lidé neustále přicházejí s novými způsoby optimalizace, jak k datům přistupovat, organizovat je, pracovat s nimi a dělat z nich různé užitečné výstupy. Od UX příruček, přes rychlejší třídicí algoritmy až po vývoj kvalitních nástrojů na tvorbu pohodlných uživatelských aplikací – to vše má lidem usnadňovat práci s informacemi a orientaci v nich.

Při představení „problému“ desítek milionů grafů, přichází řešení v podobě této bakalářské práce. Jejím účelem je implementace přívětivého a zároveň mocného uživatelského rozhraní, které i běžným lidem umožní různé manipulace s existující databází grafů. Člověk standardně pojme jen omezené množství informací a pozornost nerozloží do více než hrstky směrů. U samotného grafu lze mimo jeho vzhled identifikovat a vyjádřit desítky užitečných vlastností, pomocí nichž se dají kategorizovat a dále využívat při řešení dalších výzev a hlavolamů.

Konkrétní cíle, na které jsme se v práci zaměřili, jsou umožnit intuitivní plošné i detailní zobrazování grafů a jejich filtrování podle daných kritérií, dále jak jednotlivé, tak hromadné stahování pro pozdější práci s nimi na lokálním stroji a nakonec schopnost interakce s grafovým kalkulátorem počítajícím celou škálu vlastností grafů, který je zároveň otevřenou branou do databáze, protože originální, dosud neobsažené, grafy se jeho prostřednictvím dají přidávat do sbírky a tím rozšiřovat intelektuální hranice lidstva.

Vlastnostmi webové aplikace to však nekončí! Minimálně stejně důležitý je celý proces vývoje, proto byly pečlivě vybrány ty nejžhavější technologie, které všechno co možná nejvíce usnadní, ale zároveň nijak neuberou na kvalitě či výkonu aplikace a neomezí jeho možnosti. Obdobně podstatné nám přijde následná možnost spravování a rozšiřování aplikace.
Řekněme, že za 10 let se webové paradigma výrazně posune určitým novým směrem. K udržení kroku je vhodná flexibilita a přehlednost, kterou nám přináší v programovacím světě relativně nový, ale velmi rychle rostoucí populární frontend framework React. Skutečně pohodlný uživatelský zážitek zajišťuje předvídatelný stavový kontejner Redux a navigace na stránce se stává neskonale přirozenou díky React Router. Vývoj v těchto JavaScript knihovnách si lze jen obtížně představit bez Node.js – prostředí, které nám dovoluje spouštět JavaScript kód mimo prohlížeč – přímo na stroji. Webpack přináší chytré balení souborů pro optimalizovanou velikost a minimální počet dotazů na server, ale slouží i jako vrátka pro Babel, který náš kód překládá z modernějších až experimentálních a mocnějších verzí JavaScriptu do těch starších, kterým zase rozumí

všechny prohlížeče.

V neposlední řadě je důležité myslet i na rozšiřitelnost a vědeckou část aplikace. Cílem rozhodně není vytvořit jen rigidní blok, střípek, který se ztratí v čase, aplikaci bez budoucnosti. V nevyhnutelné situaci objevení nebo vymyšlení nových vlastností grafů je žádoucí, aby jejich integrace do databáze nijak neovlivnila chod a funkčnost webové aplikace. Zároveň v dnešní hektické informační době neustále vznikají nové formáty optimalizující velikost nebo rychlost čtení dat

Kapitola 2
Použité technologie

2.1 HTML

Téměř každý, kdo se pohybuje v technické sféře, o `HTML`[1] alespoň slyšel. HTML, neboli `Hyper Text Markup Language`, je v dnešní době dominantní značkovací jazyk na tvorbu webových stránek a aplikací. Není divu, je velmi lehký na naučení se a základy pochopí i člověk nezdatný v programování. Zároveň je relativně mocný a dají se s ním vytvořit rychle viditelné výsledky.
Vznikl v roce 1993 (25 let zpět!) a spolu s `CSS`[2] a `JavaScriptem`[3–4] tvoří neotřesitelnou triádu frontend technologií. S jeho vznikem se pojí jméno Tim Berners-Lee, který o jazyce podobném HTML uvažoval už v 80. letech minulého století.
Technologicky se vše děje tak, že tento jazyk umí interpretovat prohlížeče, které ho přečtou a následně vykreslí uživateli na obrazovku. Jednotlivé stránky mohou dostávat například ve formě souboru z místního úložiště nebo z webového serveru, který klidně běží na druhém konci světa. HTML se se stará pouze o strukturu webové stránky. Vzhled jako barvy pozadí, textu nebo jeho velikost má na starosti již výše zmíněné CSS, o kterém bude více informací níže a například uživatelskou interakci umí skvěle obstarat JavaScript, který si ale zároveň poradí i se strukturou, i se stylováním.

2.1.1 Elementy

Elementy jsou stavební bloky celé webové stránky. Prohlížeč si přečtením HTML souboru postaví stromovou strukturu `DOM`[5] popsanou níže, která se skládá právě z elementů. Ty jsou definovány tagy, které se dělí na párové a nepárové. Tagy dále mají atributy, které specifikují další jejich vlastnosti. Párové tagy obalují nějaký obsah – většinou text – a je potřeba je otevřít a uzavřít. Nepárové tagy je potřeba zavřít lomítkem na konci jejich definičního tagu.

```
<p>Odstavec</p>
```

Ukázka párového elementu odstavce, kde úvodní tag `<p>` říká, že budou následovat znaky, které vyplní obsah odstavce a vše se ukončí ve chvíli detekce ukončovacího tagu `</p>`.

```
<img src="/cesta/k/souboru.jpg" alt="ukázka kódu" />
```

Ukázka nepárového elementu obrázku, kde element nemá žádný textový obsah a uzavře se před konečnou závorkou. Atribut `src` určuje cestu k souboru a atribut `alt` definuje, co se ukáže například při selhání stažení obrázku.
Důležité atributy jsou `class` a `id`, které později JavaScriptu nebo CSS umožní zaměření na specifickou skupinu nebo na jeden konkrétní element.

```
<p class="heading-paragraph" id="first-heading">Odstavec</p>
```

2.1.2 DOM

`Document Object Model`[5] je v našem kontextu stromová struktura HTML dokumentu. Při prvním načtení stránky je definovaná daným HTML souborem, ale může se s ní manipulovat pomocí JavaScriptu – jednotlivé elementy se mohou přidávat, upravovat i odstraňovat.

Obrázek 2.1. Ukázka struktury DOM jednoduchého webu.[1]

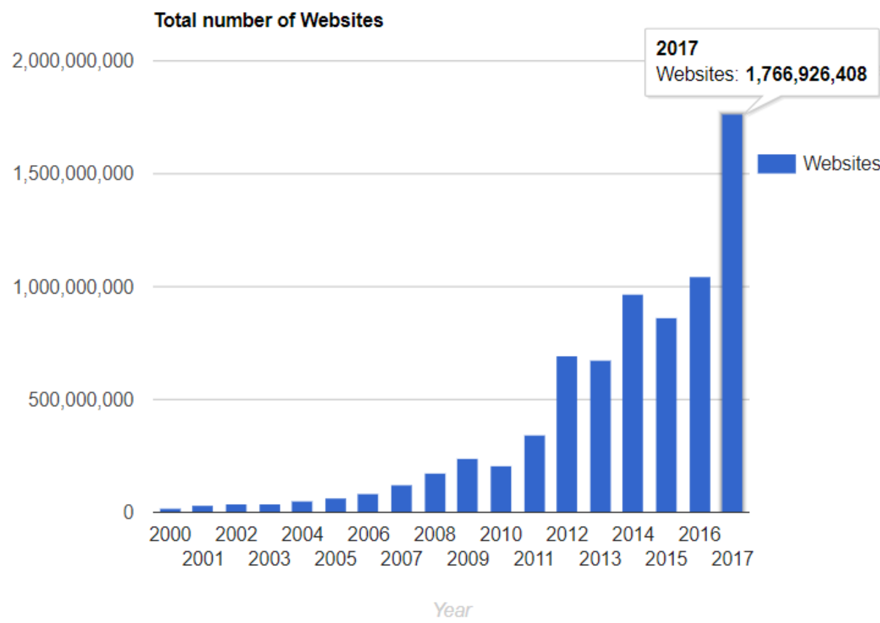
2.1.3 HTML5

Nejaktuálnější verze HTML, oficiálně vydaná v roce 2014. `HTML5`[6] přináší řadu nových elementů jako například sémantické (`<header>`, `<footer>`), grafické (`<svg>` `<canvas>`) nebo multimediální (`<audio>`, `<video>`). Dále dovoluje nové formulářové atributy (`number`, `date`, `calendar`, ...) a nová aplikační rozhraní jako geolokaci nebo lokální úložiště.
Obrovská popularita HTML zaručuje, že se lidé aktivně podílí na zlepšování jazyka a vymýšlení nových, lepších funkcionalit.

2.1.4 Alternativy

Při delší práci s technologiemi HTML a CSS člověk časem narazí na určité nuance, které ho můžou uvést do letmého zoufalství. Jedná se konkrétně o boje s rozložením stránky, s její responsivitou, skládáním elementů pod sebe nebo vedle sebe a obtížné umí být i centrování obsahu, kdy je potřeba sahat po různých nepraktických kličkách. Lidé se často diví, jak se může v roce 2018 něco takového dít a proč ještě neexistují alternativy, které by všechny tyto problémy řešily. Síla HTML a CSS je v jejich jednoduchosti a tradici. Vznikly ve svém oboru velmi brzy a nyní se těší obrovskému pokrytí.

[1] https://upload.wikimedia.org/wikipedia/commons/thumb/5/5a/DOM-model.svg/220px-DOM-model.svg.png



Obrázek 2.2. `Počet existujících webů v čase.` [7]

Ze statistiky výše je zřejmé, že i nová přelomová technologie, pokud by vůbec měla šanci, bude potřebovat hodně času, aby se těšila nějaké širší popularitě.

2.2 CSS

`Cascading Style Sheets`[2] je stylovací jazyk, který umožňuje webovým stránkám měnit svůj vzhled. Barvy, velikosti fontů, odsazení, velikosti elementů – to vše a ještě víc lze nastavit právě díky CSS. Vydáno v roce 1996, snaží se o rozdělení zodpovědností, kde HTML definuje strukturu, CSS vzhled a JS interakce.

2.2.1 Pravidla

```
p {
    font-size: 12px;
    color: redd;
}
```

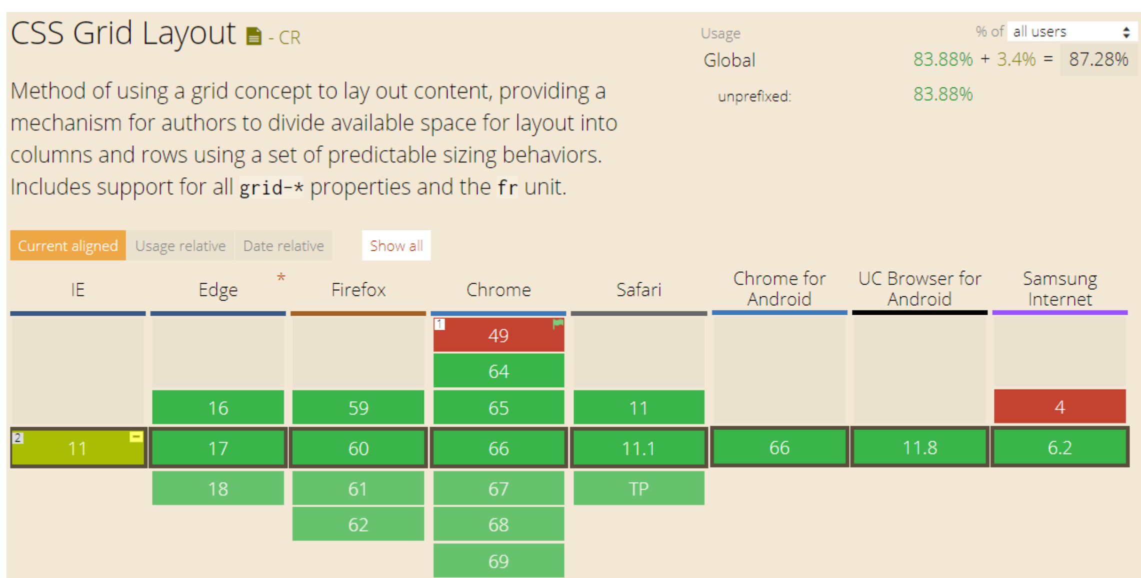
CSS kód je souborem pravidel jako to výše, které například říká, že všechny elementy odstavce na stránce budou mít velikost 12 pixelů a červenou barvu.

```
.heading, #first-heading {
    font-size: 12px;
    color: redd;
}
```

Existují takzvané `selektory`, které nestylují všechny elementy podle značky, ale specifické elementy, kterým jsme v HTML dali nějakou `class` nebo `id`. Selektory začínající `'.'` se vztahují ke třídám, `'#'` k id.

2.2.2 Kompatibilita

Běžný problém je, že existují nějaké sofistikované funkcionality, které ale nejsou podporovány všemi prohlížeči. Někdy je podpora tak malá, že nemá cenu danou věc vůbec používat v kódu, ale prohlížeče se s novými verzemi snaží integrovat i tyto nové funkcionality, a tak někdy stačí počkat pár měsíců, než podpora splní použitelnou hranici, řekněme 85% uživatelů.



Obrázek 2.3. Podpora pro grid layout ke dni 1.5.2018 [8]

Ke kontrole kompatibility je nejspolehlivější a nejpoužívanější stránkou `caniuse`[9]. Na obrázků výše je vidět reprezentace pro jednotlivé prohlížeče a jejich verze, vpravo nahoře je potom možnost vidět globální podporu uživatelů. `CSS Grid Layout` je mocný nástroj na kompozici a responsivitu webu, ale je vidět, že verze `Internet Explorer` starší než 11 ho nepodporují. Je potom na zvážení, jestli danou věc použít a přijít o některé uživatele za cenu mnohem lepšího uživatelského zážitku pro ty ostatní.

2.2.3 Knihovny

Na stylovací scénu začaly postupem času přistávat nové knihovny jako například `Bootstrap`[10] nebo `Semantic UI`[11]. Stačí si je naimportovat do projektu a od té chvíle může člověk využívat celou škálu nových možností. Většinou se jedná o chytře napsaná nebo moc hezky nastylovaná pravidla pro elementy a různé třídy. Tlačítka jsou rázem mnohem přívětivější na pohled s působivými efekty pro najetí myši nebo na kliknutí a dají se používat například chytré stylovací třídy pro kompozici elementů, kdy web vypadá dobře v prohlížeči, ale i na mobilním zařízení se vše zarovná hezky pod sebe pro pohodlný uživatelský zážitek.

2.2.4 Preprocesory

`Preprocesory`[12] jsou jazyky, které se kompilují do CSS. Na první pohled vypadají a píšou se jako CSS, ale mají mnoho funkcionalit navíc. V běžném CSS například nejsou žádné proměnné a pokud chceme stylovat naši stránku do odstínu červené barvy, musíme kód pro tuto barvu všude opakovat znovu a znovu. Když nás potom červená přejde a chceme mít web modrý, musíme ručně přepisovat spoustu řádků. Proměnná by nám práci výrazně usnadnila.

```
$color-main: red;
$font-size-large: 16px;

h1 {
    color: $color-main;
}
p {
    font-size: $font-size-large;
    color: $color-main;
}
```

po kompilaci vypadá kód takto

```
h1 {
    color: red;
}
p {
    font-size: 16px;
    color: red;
}
```

Dále například kalkukace.

```
width: 40px + 6
width: 40px - 6
width: 40px * 6
width: 40px % 6
```

Se překompiluje do

```
width: 46px;
width: 34px;
width: 240px;
width: 4px;
```

Ukázka kódu 2.1. Kód převzatý z `https://learn.shayhowe.com/advanced-html-css/preprocessors`.

Velmi užitečný je `nesting` – zanořování, který ušetří spoustu opakování se v kódu a usnadní mozku chápání zanoření jednotlivých CSS pravidel.

```
nav {
  ul {
    margin: 0;
    padding: 0;
    list-style: none;
  }

  li { display: inline-block; }

  a {
    display: block;
    padding: 6px 12px;
    text-decoration: none;
  }
}
```

Ukázka kódu 2.2. Kód převzatý z `https://sass-lang.com/guide`.

Přeložené CSS níže

```
nav ul {
  margin: 0;
  padding: 0;
  list-style: none;
}

nav li {
  display: inline-block;
}

nav a {
  display: block;
  padding: 6px 12px;
  text-decoration: none;
}
```

Ukázka kódu 2.3. Kód převzatý z `https://sass-lang.com/guide`.

Možností je opravdu mnoho. Preprocesory zpřehledňují kód, usnadňují práci s ním a rapidně roste efektivita psaní.

2.2.5 Preprocesor Sass

`Sass`[13] je preprocesor použitý v této práci. Má spolehlivou historii, aktivní údržbu i komunitu a všechny potřebné funkce.

2.3 JavaScript

`JavaScript`[3–4] se poprvé objevil v roce 1995 a v současnosti je to jeden z nejpopulárnějších a nejrychleji rostoucích jazyků na světě. Na webu se mimo interaktivity může starat prakticky o cokoliv. Běží ve všech známějších prohlížečích, takže vyzkoušet si ho může každý otevřením prohlížeče a spuštěním konzole. Při vývoji této práce je použita technologie `Node.js`[14], která dovoluje běh JavaScriptu přímo na stroji.

2.3.1 Verze JavaScriptu

JavaScript samozřejmě drží krok s dobou. Jak se mění programovací návyky a nejlepší praxe, JavaScript se přizpůsobuje a přidává nové funkce.

Year	Name	Description
1997	ECMAScript 1	First Edition.
1998	ECMAScript 2	Editorial changes only.
1999	ECMAScript 3	Added Regular Expressions. Added try/catch.
	ECMAScript 4	Was never released.
2009	ECMAScript 5	Added "strict mode". Added JSON support.
2011	ECMAScript 5.1	Editorial changes.
2015	ECMAScript 6	Many new features. Read more in JS Version 6.
2016	ECMAScript 7	Added exponential operator (**). Added Array.prototype.includes.
2017	ECMAScript 8	Added string padding. Added new Object properies. Added Async functions. Added Shared Memory.

Obrázek 2.4. Verze ECMAScript standardu, na kterém JavaScript staví.[15]

2.3.2 Babel

Problém může být, že nové funkce ještě neumí všechny verze prohlížečů, a proto vznikl nástroj Babel[16], který umí ES6+ kód překompilovat do ES5, kterému už rozumí všechny prohlížeče.

```
[1,2,3].map(n => n + 1);
// [2, 3, 4]
```

Ukázka arrow funkce, také známé jako lambda funkce z jiných funkcionálních programovacích jazyků, která se z ES6 do ES5 přeloží následovně.

```
[1, 2, 3].map(function (n) {
  return n + 1;
});
// [2, 3, 4]
```

2.3.3 Zajímavosti

Díky své relativní jednoduchosti, flexibilitě a široké škále možností je neuvěřitelně populární, a tak není divu, že vznikají knihovny, které JavaScript posouvají na další úroveň. Například díky Electronu[17] se dají s HTML, CSS a JavaScriptem tvořit desktopové aplikace na všechny platformy. React Native[18] zase pomocí JavaScriptu a Reactu produkuje mobilní aplikace na iOS i Android. Tolik možností!

2.4 Node.js

Node.js[14] je open-source runtime prostředí, které umožňuje spouštět JavaScript kód mimo prohlížeč – přímo na stroji. Je postaveno na V8 JavaScript engine od Googlu. Běží na množství platforem a dnes se těší stále větší a větší popularitě, jak na něj přecházejí různé obrovské firmy jako Netflix, Linkedin, eBay nebo NASA. Jednou z výhod Node.js je asynchronnost – zatímco PHP by nejdříve poslalo systému dotaz na soubor, počkalo by na jeho otevření, zpracování jeho obsahu, ten pak poslalo zpět a až

v tuto chvíli by bylo připravené na další dotaz, Node.js pošle systému dotaz na soubor a hned je připraveno na další dotaz od uživatele. Obsah souboru vrátí až ve chvíli, kdy je s ním systém hotový.

Node.js umí:

- Generovat dynamický obsah stránek.
- Tvořit, otevírat, číst, mazat, zavírat soubory a upravovat jejich obsah.
- Sbírat data z formulářů.
- Pracovat s databází, přidávat do ní data, mazat je a upravovat.

Primárně backendová technologie, v této práci bude Node.js využit na vývoj frontendu, s pomocí Webpacku a Babelu bude v reálném čase sledován a překládán kód pro snadné a efektivní kódování.

2.4.1 npm

Node package manager (npm)[19] je správce balíčků pro Node.js. Node.js projekty standardně ve své kořenové složce obsahují soubor package.json, který v sobě drží nějaká základní relevantní metadata jako třeba název, verzi a závislé balíčky.

```
{
  "name": "graphs",
  "version": "0.0.0",
  "scripts": {
    "compile": "webpack",
    "start": "node app.js",
  },
  "dependencies": {
    "express": "~4.15.2",
    "react": "^16.3.2",
    "react-dom": "^16.3.2",
  },
  "devDependencies": {
    "babel-core": "^6.25.0",
    "webpack": "^3.5.3",
  }
}
```

Ukázka kódu 2.4. Malá ukázka souboru ze package.json.

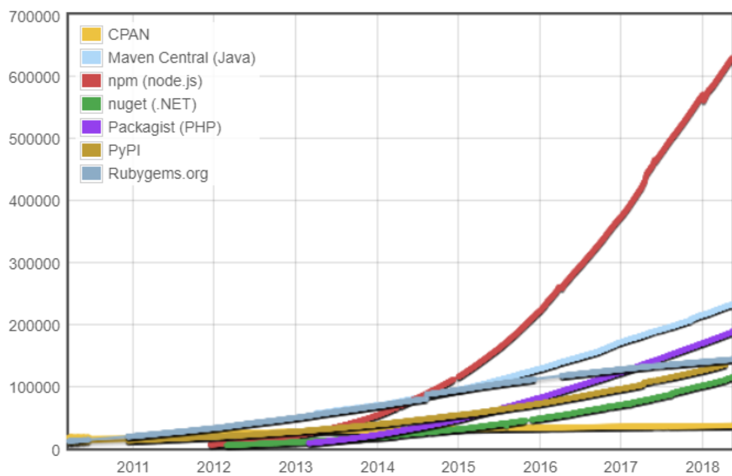
V ukázce výše lze vidět například název projektu (graphs), skripty, které můžeme spouštět zavoláním

```
npm run start
```

což spustí příkaz node app.js. A nakonec balíčky nebo knihovny, které používáme v kódu jako například express nebo react. Ty se dají do projektu stahovat a přidávat jednoduchým příkazem.

```
npm install nazev-balicku
```

To se hodí, když pracujeme například s gitem a projekt si stáhne náš kolega na vlastní počítač, kde tyto knihovny nemá. Výše zmíněným příkazem si je všechny stáhne a tím zajistí, že mu projekt bude fungovat stejně jako na našem stroji.



Obrázek 2.5. Počet existujících balíčků v čase pro různé jazyky. [20]

Je vidět, že popularita Node.js a JavaScriptu obecně rapidně roste.

2.5 React

React[21] je možná nejpopulárnější open-source frontend JavaScript knihovna současnosti. Vyvíjená a udržovaná Facebookem kolem sebe stihla od vydání v roce 2013 nasbírat hodně pozornosti a obrovskou komunitu programátorů, kteří se aktivně podílí na jejím vývoji a zdokonalování. Síla Reactu tkví v jeho jednoduchosti, efektivitě vývoje a výkonu. Výrazně usnadňuje tvorbu dynamických klientských rozhraní a chytrým překreslováním pouze potřebných částí webu ze sebe dělá mocný nástroj.

2.5.1 Virtual DOM

Operace s DOMem jako překreslování, přidávání nebo odebírání elementů ze stránky jsou celkem náročné. Hodně knihoven s tímto faktem nezachází šetrně. Pokud například máme na eshopu košík s 5 položkami a přidáme šestou, jiný framework překreslí celý košík a všech 6 věcí v něm. React pouze chytře přidá šestou. Jak to dělá? Drží si takzvaný Virtual DOM, který je pouhou reprezentací skutečného DOMu. Operace na něm jsou mnohem šetrnější a levnější na provedení než u reálného DOMu, React dokáže porovnat změny a upravit pouze to, co je potřeba.

2.5.2 JSX

JSX[22] je syntaxové rozšíření JavaScriptu. React pod pokličkou pouze operuje s DOMem pomocí JavaScriptu. Kousek JSX by mohl vypadat například takto.

```
<h1 className="greeting">Hello, world!</h1>
```

Což se pomocí Babelu přeloží do kódu, který vypadá následovně:

```
const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);
```

Tento kód vytvoří HTML element níže:

```
<h1 class="greeting">Hello, world!</h1>
```

2.5.3 Komponenty

Komponenty jsou srdce `Reactu`. Kompartmentalizace kódu a jeho rozdělení do znovupoužitelných kousíčků přináší úžasnou přehlednost a znovupoužitelnost. Komponenta je většinou nějaký malý kousek UI, který se dá vkládat na různá místa. Například patička stránky nebo kartička s profilem uživatele. Nemusíme je všude vkládat a upravovat manuálně, ale všude používáme jejich komponentu a pokud ji chceme změnit, děláme to jen na jednom místě – tam, kde byla definována. Dají se všemožně kombinovat a zanořovat. Předávají se jim takzvané `props`, které uvnitř nich dynamicky upravují jejich stav nebo co mají produkovat. Jsou velmi důležité a jedná se o objekt – argument, který dostávají odněkud seshora. Můžou v něm být libovolné informace a přistupuje se k nim přes `this.props`. U jednoduchých funkcionálních komponent jako ta níže pouze přes `props`.

```
// definice komponenty
const Greetings = props => <h1>Hello, {props.name}!</h1>

// vykreslení
ReactDOM.render(
  <Greetings name="World" />,
  document.body
);
```

kód výše nám do těla stránky vykreslí nadpis říkající: `Hello, World!`.

2.5.4 Stav a event handlery

Stav aplikace drží všechen dynamický obsah, který potom stránka zobrazuje. Komponenty ho berou a v podstatě jen promítají do požadované formy. Stav mohou mít pouze takzvané chytré komponenty, které se definují následujícím způsobem.

```
class Counter extends React.Component { ... }
```

Následuje příklad komponenty `Counter`, která ukazuje počítadlo, které se inkrementuje po kliku na číslo.

```
class Counter extends React.Component {
  // implicitní stav při načtení komponenty
  state = {
    counter: 0
  }

  // funkce starající se o inkrementaci počítadla
  clickHandler = () => {
    this.setState({ counter: this.state.counter + 1 })
  }

  // funkce, která definuje co se vykresluje
  render() {
    return (
      // Event handler navěšený na kliknutí na <div>
      <div onClick={this.clickHandler}>
        {this.counter}
      </div>
    );
  }
}
```

V této práci stav bude řešen přes knihovnu Redux

2.5.5 Lifecycle metody

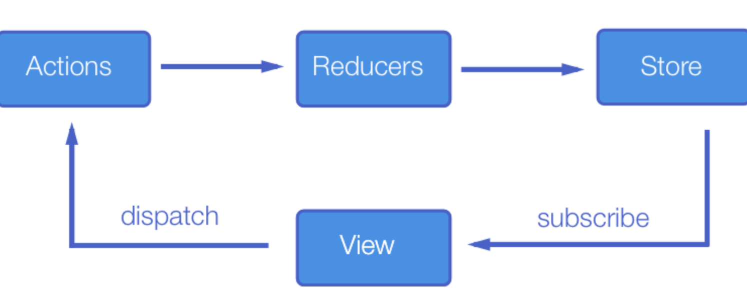
Lifecycle metody[23] jsou esenciální součástí Reactu. Dají se definovat pouze u chytrých komponent a s jejich pomocí můžeme například stahovat data ihned po načtení stránky, reagovat na změny stavu specifickým způsobem nebo optimalizovat celou komponentu určením, kdy ji překreslovat a kdy ne. Příklad metody componentDidMount

```
class UserList extends React.Component {
  state = {
    users: []
  }

  componentDidMount() {
         fetch('/api/users')
            .then(res => res.json())
            .then(result => this.setState({ users: result.users }))
  }

  render() {
    return (
      <div>
        {this.state.users.map(user => <p>{user.name}</p>}
      </div>
    );
  }
}
```

Kód výše po načtení komponenty zavolá dotaz na uživatele, po vrácení výsledku do stavu uloží všechny uživatele a vykreslí jejich jména.
V této práci je napsáno pouze asi 15 řádek pravého HTML a to v souboru public/index.html. Zbytek je JSX z Reactu, tedy obsah vygenerovaný JavaScriptem.

2.6 Redux

Redux[24] je stavový kontejner pro tuto práci. Jedná se o jediný zdroj informací nad celou aplikací, což znamená, že stav není rozesetý po různých místech v aplikaci a komponentách. Dá se k němu dostat odkudkoliv a výhodou oproti implicitnímu stavu Reactu je například to, že po odchodu na jinou stránku a návratu zpět nám data nezmizí, ale tabulka výsledků zůstane zapopulovaná. V případě implicitního stavu by nám data zmizela ve chvíli odpojení komponenty z DOMu.



Obrázek 2.6. Architektura Reduxu[1]

Na obrázku výše je vidět jednosměrný tok informací. `View` – v našem případě nějaká komponenta nebo stránka je napojená na `Store`, který jí poskytuje stavová data a `akce`. `Akce` jsou balíčky informací, které posílají data z aplikace do `Storu` a jsou jeho jediným zdrojem informací. Jedná se o obyčejné `JavaScriptové` objekty. Musí obsahovat string `type` a libovolnou informaci k tomu.

```
{
  type: 'SET_OFFSET',
  offset: 5,
}
```

Důležité jsou `tvořiče akcí`, což jsou funkce, které vrací objekt dané `akce`

```
const setOffset = offset => ({
  type: 'SET_OFFSET',
  offset: offset
})
```

`Akce` se dají vyvolat například kliknutím na tlačítko, tím se zavolá `tvořič akce` obalený funkcí `dispatch`, která umožňuje odesílání `akcí` do `Storu`, kde je podle jejich `typu` odchytí příslušný `Reducer` a adekvátně upraví `Store`.
`React` komponenty se napojují na `Store` pomocí funkce `connect` z knihovny `react-redux`, která přijímá dva argumenty. První je funkce `mapStateToProps`, která definuje, jak se transformuje `Store` neboli stav aplikace do `props` pro danou komponentu. Druhý je funkce `mapDispatchToProps`, která obalí `tvořiče akcí` funkcí `dispatch`, čímž umožní vysílat `akce` do `Storu`.

```
import { setOffset } from './actions'

const mapStateToProps = state => ({
    offset: state.offset
})

const mapDispatchToProps = {
    setOffset
}

connect(mapStateToProps, mapDispatchToProps)(Pagination)
```

[1] https://cdn-images-1.medium.com/max/1600/0*95tBOgxEPQAVq9YO.png

Ukázka napojení komponenty `Pagination` ke `Storu`. Nyní z `props` bude moct používat proměnnou `offset` a volat funkci `setOffset`.

2.7 React Router

`React Router`[25] do této práce přinese dynamické routování. Starší a obecně známe je routování statické, to znamená, že po dotazu na adresu například `www.stranka.cz/produkty` server zpracuje dotaz a pro cestu `produkty` vrátí korespondující `.html` soubor se seznamem produktů. V dynamickém routování však existuje pouze jeden `.html` soubor, který server vrací pro libovolnou cestu a `JavaScript` v aplikaci až následně pozná, kde se uživatel nachází a vykreslí odpovídající stránku. Pro pohyb v aplikaci pak slouží odkazy stejně jako na běžném webu, ale již ne nedělá další dotaz na celou stránku, což značně ulehčuje práci serveru, nýbrž se pomocí `JavaScriptu` pouze překreslí pohled.

2.8 Webpack

`Webpack`[26] se v této práci stará o spoustu věcí a nesmírně usnadňuje vývoj. Obecně se jedná o `static module bundler`, což znamená, že projde celý kód, podívá se na vzájemné závislosti, potřebné transpilace kódu a vše úhledně zabalí do optimalizovaných balíčků. V praxi to znamená, že projekt může mít desítky `.js` a `.scss` souborů, webpack je projde, přeloží verze `JavaScriptu` z `ES6+` do `ES5`, přeloží `Sass` soubory do normálního `CSS` a vyplivne jediný minifikovaný `.js` a `.css` soubor, což urychluje načítání stránky u klienta, protože nemusí dělat desítky různých dotazů, ale jen dva.

2.9 Git

`Git`[27] je open-source verzovací nástroj. Celkem jednoduchý s širokou škálou možností dovoluje týmu programátorů z různých konců světa spolupracovat na jednom a tom samém projektu téměř v reálném čase. V této práci byl použit, protože zároveň s tvorbou frontendu probíhal i vývoj backendu a `git` velmi pomohl s vývojem, integrací a testováním nových věcí. Zaručil, že se s kolegy díváme na stejný kód, aniž bychom byli ve stejné místnosti.

Kapitola 3
Vlastní implementace

K dispozici je vzdálený server na adrese `graphs.felk.cvut.cz`, na kterém sídlí data aplikace v produkci. Pro vývoj bylo potřeba začít `git` repositářem, který je k nalezení na `https://gitlab.fel.cvut.cz/graphs/development`. Pro hlavní vývoj se používá větev `master`, produkční server běží na větvi `prod`. Webové aplikaci je vyhrazena složka `node-www`. Projekt nezačínal stavět na žádných existujících aplikacích, a tak nezbývalo než se pustit do práce inicializací `Node.js` projektu.

3.1 Prostředí Node.js

V kořenové složce `node-www` je k nalezení soubor `package.json`, který vznikl zavoláním příkazu

```
npm init
```

a vyplněním základních informací. Je v něm název projektu, jeho verze a různé spouštěcí skripty, starající se o kompilaci projektu nebo puštění serveru. Dále obsahuje všechny závislosti, které pomáhají při vývoji a starají se o chod aplikace. K tomu byl použit příkaz

```
npm install --save react react-dom react-redux webpack ... (a další)
```

Následně stačí nastavit ostatní nástroje jako `Webpack` a přepsat správně spouštěcí skripty, které lze volat příkazem

```
npm run nazev-prikazu
```

takže například v produkční verzi jsou k nalezení skripty:

```
"scripts": {
    "compile": "webpack",
    "start": "NODE_ENV=production node ./app.js"
},
```

aktualizace produkčního serveru tedy proběhne tak, že se změny stáhnou z `gitu` příkazem

```
git pull origin master
```

projekt se zkompiluje přes webpack zavoláním

```
npm run compile
```

a následně spustí

```
npm start
```

Po zprávě `'Listening on...'` jsou změny již aktivní a dostupné na adrese `http://graphs.felk.cvut.cz/`.

3.1.1 Struktura projektu

Nejzajímavější věci z frontendového hlediska se odehrávají primárně ve složce `src/graphs`, ve složce `src` je také k nalezení soubor `config.js`, který má na starosti správu všech vlastností grafu, neboli také sloupečků v databázi. V případě manipulace s nimi, jejich typem nebo po přidání či ubrání nové vlastnosti je potřeba upravit tento soubor – více o něm v sekci `Další rozšiřování`.
Složka `src/graphs` má následující strukturu:

```
src/graphs
  |-components ............ prezentační React komponenty
  |-containers ............ chytré React komponenty napojené na Redux
  |-lib ................... pomocné funkce a knihovny
  |-pages ................. komponenty všech stránek
  |-redux ................. všechny Reduxové soubory
  |-App.js ................ primární komponenta aplikace
  |-App.scss .............. hlavní styly aplikace
  |-fonts.scss ............ fonty
  |-index.js .............. místo napojení React aplikace do HTML
  |-minframework.min.css .. minimalistický CSS framework
  |-routes.js ............. soubor s dosažitelnými stránkami
  |-variables.scss ........ CSS proměnné
```

3.2 Webpack

`Webpack` je používaný na produkci ke kompilaci statických souborů a transpilaci `JavaScript` kódu do starších verzí.

```
const path = require('path');
const webpack = require('webpack');

module.exports = {
  context: path.join(__dirname, 'src'),
  entry: {
    graphs: './graphs/index.js',
  },
  output: {
    path: path.join(__dirname, 'public'),
    filename: '[name].bundle.js',
  },
  module: {
    rules: [
      {
        test: /\.js$/,
        exclude: /node_modules/,
        use: 'babel-loader',
      },
      {
         test: /\.(eot|ttf|woff|woff2)$/,
         use: 'file-loader?name=public/fonts/[name].[ext]'
       },
      {
        test: /\.scss$|\.css$/,
        use: ExtractTextPlugin.extract({
          fallback: 'style-loader',
          use: ['css-loader', 'resolve-url-loader', 'sass-loader']
        })
      }
    ],
  },
};
```

Ukázka kódu 3.1. Ukázka zjednodušeného souboru `webpack.config.js` z této aplikace.

Důležité jsou body `entry`, `output` a `rules`.
`Entry` říká, co je vstupní soubor. `Webpack` si ho přečte a následně do hloubky projde všechny potřebné závislosti. To znamená, že pokud soubor `index.js` importuje `App.js`, který importuje `HomePage.js`, `webpack` všechno toto projde, přeloží kód a zoptimalizuje ho do jednoho velikého souboru.
`Output` specifikuje, kam se bude ukládat výsledek celého procesu. V našem případě je to `public` složka, ze které potom server klientovi vrací `index.html` soubor, dále příslušné `.css` a `.js` soubory nebo obrázky.
`Rules` definuje pravidla, která `webpack` zkontroluje. V kódu výše jsou vidět celkem tři. To první podle regulárního výrazu v atributu `test` kontroluje všechny `.js` soubory a podsouvá je `babel-loaderu`, knihovně získané z `npm`, která transpiluje nové funkce kódu do starších, kompatibilních verzí pro všechny prohlížeče. Druhé pravidlo se stará o soubory `fontů` a třetí kontroluje všechny `.scss` a `.css` soubory. `Sass` syntax se přeloží do běžného `CSS` a vše se potom zminifikuje do jednoho souboru.

3.2.1 Produkce

V produkci stačí zavolat příkaz `webpack`. `Webpack` si přečte konfigurační soubor `webpack.config.js` v kořenové složce a provede příslušné úpravy. V tomto případě všechno zkompiluje podle pravidel zmíněných výše a ve chvíli, kdy klient udělá dotaz na server, vrátí se mu `index.html` soubor, který už si stáhne potřebné `assety`.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="initial-scale=1.0">
    <link rel="stylesheet" type="text/css" href="/graphs.css">
    <title>Graphs</title>
  </head>
  <body>
    <div id="root"></div>
    <script src="/vendor.bundle.js"></script>
    <script src="/graphs.bundle.js"></script>
  </body>
</html>
```

Ukázka kódu 3.2. Soubor `public/index.html`.

3.2.2 Vývoj

Při vývoji je používán `webpackDevMiddleware`, který umožňuje vývoj bez zbytečného čekání. Místo toho, aby bylo po každé změně kódu potřeba zavolat `webpack`, který by chvíli kompiloval kód a až následně by byl použitelný, můžeme použít tento nástroj a dostaneme mnohem efektivnější funkcionalitu. Spuštěním aplikace s tímto middlewarem se `webpack` připraví, zkompiluje kód a čeká na změny, které okamžitě servíruje. Stačí uložit soubor například stiskem kombinace kláves `CTRL+S`, `webpack` detekuje změnu a překompiluje jen požadovanou část kódu bez zbytečného čekání. V prohlížeči se po stisku `F5` změny téměř bez prodlevy projeví.

```
const webpackDevMiddleware = require('webpack-dev-middleware')
const webpackConfig = require('./webpack.config.js')
const webpack = require('webpack')
const express = require('express')
const app = express()
const compiler = webpack(webpackConfig)

// webpack
app.use(webpackDevMiddleware(compiler, {
  hot: true,
  filename: 'bundle.js',
  publicPath: '/',
  stats: {
    colors: true,
  },
  historyApiFallback: true,
}))
```

Ukázka kódu 3.3. Ukázka ze souboru `app.js`.

3.3 React

`React` se do aplikace inicializuje v souboru `src/graphs/index.js`.

```
import React from 'react'
import ReactDOM from 'react-dom'
import { Provider } from 'react-redux'
import { createStore, applyMiddleware } from 'redux'
import logger from 'redux-logger'
import thunk from 'redux-thunk'
import { BrowserRouter as Router, Route } from 'react-router-dom'

import reducer from './redux'
import App from './App'

const store = createStore(reducer, applyMiddleware(thunk, logger))

ReactDOM.render(
  <Provider store={store}>
    <Router>
      <Route path="/" component={App} />
    </Router>
  </Provider>,
  document.getElementById('root')
)
```

Ukázka kódu 3.4. Soubor `src/graphs/index.js`.

Ve chvíli, kdy se v prohlížeči načte `index.html`, proběhne dotaz na tento `JavaScriptový` soubor, který se po načtení zavolá a v něm funkce `ReactDOM.render()`, která do prvku s `id='root'` přítomného v `index.html` vykreslí celou aplikaci. Nejdříve vše obaluje komponenta `Provider`, která je z `Reduxu` a poskytuje stav aplikace, o vrstvu níže je vidět `Router`, který se stará o routování uvnitř aplikace a nakonec už se vykresluje soubor `App.js`. Vše před voláním `ReactDOM.render()` jsou potřebné importy a konfigurace například `Redux Storu` a jeho middlewaru jako například logování akcí.
Ve zbytku aplikace je spousta komponent a stránek, kde se řeší konkrétní aplikační logika, grafika, stylování, kompozice a funkcionalita webu. Většina kódu vypadá jako běžné `HTML`, které je však okořeněné o dynamiku a všemožné `JavaScript` vychytávky.

3.4 Image.js

Pro zajímavost a ukázku možností se podíváme na komponentu `Image.js`

```
import React from 'react'
import PropTypes from 'prop-types'

import './Image.scss'

const engineOptions = ['neato', 'dot', 'fdp', 'circo', 'twopi']

class Image extends React.Component {
  state = {
    engine: 'neato',
  }

  render() {
    const { g6 } = this.props
    const { engine: currentEngine } = this.state
    const url = `/api/image/gen?g6=${encodeURIComponent(g6)}&engine=
                ${currentEngine}`

    return (
      <div className="image">
        <img src={url} alt={`graph six ${g6}`} />
        <select
          value={currentEngine}
          onChange={(e) => this.setState({ engine: e.target.value })}
          className="engine-options"
        >
          {engineOptions.map(engine => (
            <option
              value={engine}
              className={`engine-options__option
              ${currentEngine === engine ? 'active' : ''}`}
            >{engine}</option>
          ))}
        </select>
      </div>
    )
  }
}

Image.propTypes = {
  g6: PropTypes.string.isRequired,
}

export default Image
```

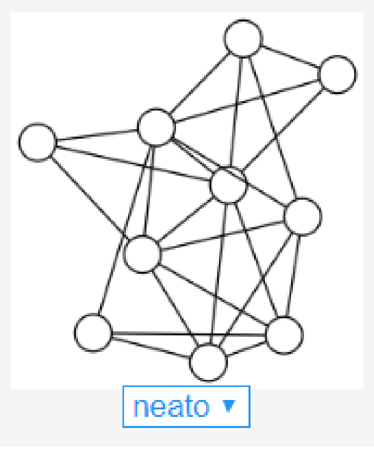
Ukázka kódu 3.5. Soubor `src/graphs/components/Image.js`.

Jelikož se jedná o soubor využívající `React`, importujeme ho. `PropTypes` je pomocná knihovna, která přináší formu typování do normálně slabě typovaného `JavaScriptu`. Na konci souboru si můžeme definovat `propTypes` dané komponenty, abychom určili, jakého typu očekáváme `props`, které dostane a zda jsou požadované nebo ne. Pokud bychom někde zavolali `<Image />` komponentu bez `g6`, dostaneme v konzoli prohlížeče upozornění a můžeme vše snadno opravit.

Dále se importuje `.scss` soubor, který se stará o stylování komponenty, definuje třeba maximální šířku obrázku a barvu text `<select>` elementu. `const engineOptions` definuje v současné chvíli dostupné enginy na generaci obrázků. Pokud přibude nějaký nový, stačí ho přidat například na konec pole a vše bude fungovat, jak má.
Pokračuje se definováním samotné `Image` komponenty. Jelikož chceme u konkrétního obrázku hlídat jeho vybraný engine, použijeme lokální stav komponenty, potřebujeme proto extendovat `React.Component`. Na dalším řádku se definuje stav na implicitní hodnotu enginu: `neato`. Následuje `render()` metoda, která musí být přítomna v každé `React.Component` komponentě, protože říká, co se bude nakonec vykreslovat. Uvnitř ní používáme proměnnou `g6` z `props` a `currentEngine` ze stavu. Dynamicky konstruujeme požadované url, na které se budeme tázat na obrázek pro požadovaný graf a engine.
Vracíme `<div>` obalující `<img>` s vygenerovaným url v `src` atributu a `<select>`, kterým si můžeme vybírat konkrétní engine. Současná hodnota `<select>` elementu se bere ze stavu a `onChange` event handler při změně zavolá metodu `this.setState()`, do které pošle nový stav. Změna se ihned projeví, proběhne dotaz na obrázek s novým url a upraví se současný výběr v `<select>` elementu.
Použití komponenty:

```
<Image g6="IACh[^xtw" />
```

A výsledek na webu:



Obrázek 3.1. Komponenta Image

3.4.1 CompletenessPage.js

Dále si ukážeme chytrou komponentu s lifecycle metodou, napojenou na `Redux`.

```
import React from 'react'
import PropTypes from 'prop-types'
import { connect } from 'react-redux'

import { graphColumns } from '../../config'
import { Pagination } from '../components'
import { fetchCompleteness } from '../redux/completeness/actions'
import './CompletenessPage.scss'

class CompletenessPage extends React.Component {
  componentDidMount() {
      this.props.fetchCompleteness({ limit: 100, offset: 0 })
  }

  render() {
     const { results, fetchCompleteness, meta } = this.props
     return (
        <div className="completeness-page">
             <div className="container">

             ...obsah stránky...

             <Pagination meta={meta} fetch={fetchCompleteness}/>

             </div>
        </div>
        )
    }
}

CompletenessPage.propTypes = {
    meta: PropTypes.object.isRequired,
    results: PropTypes.array.isRequired,
    fetchCompleteness: PropTypes.func.isRequired,
}

export default connect(({ completeness }) => ({
    results: completeness.all,
    meta: completeness.meta,
}), { fetchCompleteness })(CompletenessPage)
```

Ukázka kódu 3.6. Soubor `src/graphs/pages/CompletenessPage.js`.

O importech a `propTypes` jsme již mluvili, a tak se podívejme na `lifecycle` metodu `componentDidMount`, ta se zavolá vždy, kdy je komponenta použita a nasazena do pohledu. Je to ideální místo na natažení nových informací. V tomto konkrétním případě se jedná o stránku s informacemi o kompletnosti grafů v naší databázi. Po příchodu na stránku je chceme uživateli zobrazit, a tak si je nejdříve musíme stáhnout dotazem do databáze. O to se stará funkce `fetchCompleteness`, které přijímá objekt s parametry `limit`, který určuje, kolik výsledků chceme, a `offset`, kolik jich chceme přeskočit kvůli stránkování. Tu rozeberu v `Sekci 4 - Komunikace s backendem`.
Dále je zajímavá komponenta `<Pagination>`, které se přes `props` předává funkce na

natažení nových výsledků a `meta` objekt, který obsahuje `limit`, `offset` a `count` – tedy kolik dat stahovat na jednu stránku, na kolikáté stránce zrovna jsme a celkový počet dat. Komponenta už se sama postará o vykreslení příslušného `HTML` a zajistí funkcionalitu stahování správných dat. Je použitelná na libovolné stránce, kde chceme stránkovat data. Takto je použita i na stránce s výsledky všech grafů.

Obrázek 3.2. Komponenta Pagination -- stránkování

Za zmínku stojí i `connect` na konci souboru. V prvním argumentu přijímá `Store` aplikace, ze kterého používá pouze `completeness Reducer`. Do `props.results` mapuje všechny výsledky a do `props.meta` meta objekt pro `<Pagination>`. Ve druhém argumentu dává do `props` funkci `fetchCompleteness` obalenou `dispatchem`, takže je napojená na `Store`.

3.5 Redux

Všechny soubory spojené s `Reduxem` v této aplikaci jsou umístěny ve složce `src/graphs/redux`. V souboru `index.js` se kombinují jednotlivé `Reducery`, které se v první řadě dělí kvůli přehlednosti a jednodušší imutabilitě kódu. Důležité jsou potom jednotlivé složky, kde lze vždy najít soubor `actions.js` a `reducer.js`. Ukážeme si zjednodušené verze ze složky `computer`.

```
export const SET_G6 = 'SET_G6'
export const SET_MATRIX = 'SET_MATRIX'

export const setG6 = g6 => ({
  type: SET_G6,
  payload: { g6 }
})

export const setMatrix = matrix => ({
  type: SET_MATRIX,
  payload: { matrix }
})
```

Ukázka kódu 3.7. Ukázka ze souboru `src/graphs/redux/computer/actions.js`.

Na vrchu souboru se definují konstanty, které se používají v `typu` u `tvořičů akcí` a taky v `reducerech` při odchytávání akcí. Předejde se tímto překlepům ve stringu a zbytečným chybám, jelikož se to používá na více místech.
Dále se definují `tvořiče akcí` – funkce, které vrací objekt `akce` s typem a nějakými daty.

```
import * as actions from './actions'

const defaultState = {
  g6: '',
  matrix: '',
}

const resultsReducer = (state = defaultState, action = {}) => {
  const { type, payload } = action

  switch (type) {
    case actions.SET_G6: {
      return {
        ...state,
        g6: payload.g6.trim(),
      }
    }

    case actions.SET_MATRIX: {
      return {
        ...state,
        matrix: payload.matrix,
      }
    }

    default:
      return state
  }
}

export default resultsReducer
```

Ukázka kódu 3.8. Ukázka ze souboru `src/graphs/redux/computer/reducer.js`.

`Reducer` odchytává `akce` podle typu a vrací adekvátně upravený stav. V případě vyvolání `akce` `setG6` s parametrem `'E???'`, protéká typ `SET_G6` `Reducerem`, tam se odchytí ve `switchi` a vrátí hodnoty původního stavu s upraveným atributem `g6` na hodnotu `'E???'`.

3.6 React Router

Pro fungování `React Routeru` bylo potřeba v souboru `src/graphs/index.js` obalit aplikaci komponentou `<Router>`, ta jí dodává informace jako na které `routě` se právě uživatel nachází, historii procházení a další.

```
render() {
    return (
      <div className="app">
        <Route path="/" component={Header} />
        <Route path="/" component={Notification} />
        <div className="page-wrapper">
          <Switch>
            {routes.map(({ path, component, exact }) =>
              <Route key={path} path={path} exact={exact}
                component={component} />)}
            <Route component={NotFoundPage} />
          </Switch>
        </div>
      </div>
    )
  }
```

Ukázka kódu 3.9. Ukázka kódu ze souboru `src/graphs/App.js`

Komponenta `<Route>` se vykreslí vždy, kdy se `prop path` shoduje se začátkem současného `url` za lomítkem uživatele. Pokud je `prop exact` nastavena na `true`, vykreslí se jen při absolutní shodě. Je vidět, že `<Header>` a `<Notification>` komponenty se vykreslují vždy, protože lomítko je přítomné implicitně. O něco níže je komponenta `<Switch>`, která vykreslí pouze první `<Route>`, se kterou proběhne shoda. Uvnitř `<Switch>` komponenty se mapují `routy` ze souboru `src/graphs/routes.js`, kde jsou definované dosažitelné stránky:

```
import {
  HomePage,
  AboutPage,
  DetailPage,
  ComputerPage,
  CompletenessPage,
} from './pages'

const routes = [
  {
    path: '/about',
    component: AboutPage,
  },
  {
    path: '/complete',
    component: CompletenessPage,
  },
  {
    path: '/computer',
    component: ComputerPage,
  },
  {
    path: '/',
    component: HomePage,
    exact: true,
  },
  {
    path: '/detail/:g6',
    component: DetailPage,
  },
]

export default routes
```

Ukázka kódu 3.10. Ukázka kódu ze souboru `src/graphs/routes.js`

Pokud neproběhne shoda, vykreslí se `404` stránka, znamená to, že uživatel je na nedefinované `url`.

3.7 Css

Je použit preprocesor `Sass` a o zbytek už se stará nastavený `webpack`. Konkrétně `webpack` projde všechny soubory s příslušnými koncovkami a správně vše zabalí do `public` složky. V aplikaci se používá `BEM`, což je způsob psaní stylů, který je organizovanější a srozumitelnější než standardní `CSS`. `BEM` znamená `block`, `element`, `modifier` a jeho pravidlo by mohlo vypadat například takto:

```
.person__hand--left
```

`Block` je nějaký větší celek, ve kterém pracujeme, třeba `header`. `Element` je důležitá část celku, například `button` a `modifier` je nějaká vlastnost elementu, třeba `active`. `CSS` třídy jsou potom delší, což přináší více psaní, ale ve chvíli, kdy kód čte někdo další, mnohem rychleji pochopí k čemu konkrétní třída slouží.

3.7.1 Fonty

Pro příjemný vzhled byl použit volně dostupný font `Roboto`. Jeho soubory jednotlivých formátů jsou ve složce `public/fonts` a definice pro importování do aplikace, které umožní používání jsou v souboru `src/graphs/fonts.scss`. Následuje ukázka ze souboru, kde je ukázka použití `@font-face` pravidla z `CSS`.

```
@font-face {
    font-family: 'Roboto';
    src: url('../../public/fonts/roboto-regular.eot');
    src: url('../../public/fonts/roboto-regular.woff') format('woff'),
         url('../../public/fonts/roboto-regular.woff2') format('woff2'),
         url('../../public/fonts/roboto-regular.ttf') format('truetype');
    font-weight: normal;
    font-style: normal;

}
```

Ukázka kódu 3.11. Ukázka kódu ze souboru `src/graphs/fonts.scss`

3.7.2 Scss

U stylovacích souborů komponent je důležité nezapomenou na import, takže například v souboru `App.js` je potřeba někde nahoře mít

```
import './App.scss'
```

čímž aplikace ví, že se tento stylovací soubor používá a může ho zahrnout do finálního `buildu`.

Kapitola 4
Komunikace s backendem

Aplikace spolupracuje s paralelně vyvíjeným backendem[28]. `API` poskytuje několik `endpointů`, o kterých se dá více dočíst v dokumentaci na adrese `https://gitlab.fel.cvut.cz/graphs/development/blob/master/node-www/src/docs/docs.md`. Backend běží na knihovně `Express.js`[29] a vrací odpovědi ve formátu `json`.

4.1 Fetch API

`Fetch API`[30] je rozhraní pro tvoření síťových dotazů. V nových verzích prohlížečů `Google Chrome`[31] a `Firefox`[32] je funkce `fetch` globálně přístupná. Například `IE11`[33] ji však nepodporuje, a tak bylo potřeba použít `polyfill`[34].
Nejzákladnější použití funkce `fetch`[35] je, že přijme `url` jako argument a vrátí `promise`.

4.1.1 Promise

`Promises`[36] jsou objekty reprezentující eventuální splnění nebo selhání asynchronní operace. Dají se na ně volat metody `.then()` a `.catch()`.
`.then()` se vykoná v případě, že byla požadovaná operace úspěšná, do `.catch()` bloku se přejde v případě neúspěchu.

4.1.2 Použití v aplikaci

Níže rozebereme jeden příklad použití funkce `fetch` v `Reduxové akci` a co se následně stane v `reduceru`. Navazujeme na `Sekci 3.4.1 - CompletenessPage.js`, kde jsme v lifecycle metodě `componentDidMount` zavolali

```
this.props.fetchCompleteness({ limit: 100, offset: 0 })
```

Nyní se podíváme do souboru s definicí akce.

```
export const fetchCompleteness = ({ limit, offset }) =>
                                            (dispatch, getState) => {
  dispatch(createAction(FETCH_COMPLETENESS_START, offset))
  const url = `/api/complete?limit=${limit}&offset=${offset}`

  fetch(url)
    .then(response => {
      if (response.ok) return response.json()
      return Promise.reject(response.json())
    })
    .then(payload =>
          dispatch(createAction(FETCH_COMPLETENESS_SUCCESS, payload)))
    .catch(err => {
      err.then(e => {
        dispatch(notify(`Fetching completeness error: ${e.message}`),
                                            5000)
        dispatch(createAction(FETCH_COMPLETENESS_ERROR, e))
      })
    })
}
```

Ukázka kódu 4.1. Kus kódu ze souboru `src/graphs/redux/completeness/actions.js`

Ve chvíli zavolání funkce se vyšle `akce` s typem `FETCH_COMPLETENESS_START` a `offsetem` definujícím současnou stránku. Na `akci` reagujeme v `reduceru` nastavením `loading: true` a upravením `meta` objektu současným `offsetem`.

```
case actions.FETCH_COMPLETENESS_START: {
      return {
        ...state,
        meta: { ...state.meta, offset: payload },
        loading: true,
      }
    }
```

Ukázka kódu 4.2. Část kódu ze souboru `src/graphs/redux/completeness/reducer.js`

Dále se dynamicky nastavuje `url` na endpoint s daty a parametry určujícími, kterou část dat budeme chtít. Následně se s touto `url` volá funkce `fetch`. Samotné volání vrací `promise` objekt, tedy jakýsi příslib dat. V případě, že server dotaz zpracuje a vrátí nějakou odpověď, která může být úspěšná i neúspěšná, v sobě `fetch` zavolá `Promise.resolve()` s daty odpovědi. Tím vrací nový připravený `promise`, který odchytává první volání `.then()`. Tam se kontroluje, zda má odpověď `ok` atribut, což znamená, že se vrátila požadovaná data. Pokud ne, na dotaz například nemáme požadovaná práva a server nás o tom informuje, v takovém případě se vrací `Promise.reject()` s `json` daty a funkce pokračuje ve vyhodnocování až v `.catch()` bloku. Pokud byla odpověď `ok`, vrací se odpověď, na kterou se zavolá `.json()`, což z ní dostane pouze pro nás zajímavá `json` data. Pokračuje se do dalšího `.then()` bloku, kde se vysílá `akce` s typem `FETCH_COMPLETENESS_SUCCESS` a `json` daty ze serveru. Ta se v `Reduxu` zpracovává následovně:

```
case actions.FETCH_COMPLETENESS_SUCCESS: {
      return {
        ...state,
        all: payload.rows,
        meta: { ...state.meta, count: payload.count },
        loading: false,
      }
    }
```

Ukázka kódu 4.3. Část kódu ze souboru `src/graphs/redux/completeness/reducer.js`

Vrací se nový stavový objekt, kde se nastavuje `loading: false`, protože data se načetla, dále se do `all` nastaví data z odpovědi podle definice z `api` a do `meta` objektu se nastaví jejich celkový `count` – počet pro stránkování. Data jsou nyní v `Redux storu`. Komponenta `CompletenessPage.js` je na něj napojená `Redux` funkcí `connect`

```
export default connect(({ completeness }) => ({
    results: completeness.all,
    meta: completeness.meta,
}), { fetchCompleteness })(CompletenessPage)
```

data se v ní okamžitě aktualizují. Komponenta si data ve své `render()` funkci vytahává z `props` a pole výsledků pomocí funkce `.map()` zobrazuje uživateli jako `HTML`.

```
render() {
    const {results, fetchCompleteness, meta} = this.props
    return (
        ...
        {results.map((r, i) => (
            <div key={i} className="results__row">
                  ....
           </div>
        ))}
        ...
    )
```

V případě, že se nepodařilo kontaktovat server, došlo k selhání v databázi nebo došlo na backendu k jiné chybě, vyhodnocuje se `.catch()` blok. V něm se vyšlou dvě `akce`. První je notifikace, která informuje uživatele o chybě i s textem z odpovědi, například `'Something broke in the DB'`. Nakonec se vysílá `akce` s typem `FETCH_COMPLETENESS_ERROR` a chybou, což se v `reduceru` vyhodnotí už jenom tak, že se nastaví `loading: false`. Uživatel již byl o chybě informován notifikací.

```
case actions.FETCH_COMPLETENESS_ERROR: {
      return {
        ...state,
        loading: false,
      }
    }
```

Ukázka kódu 4.4. Část kódu ze souboru `src/graphs/redux/completeness/reducer.js`

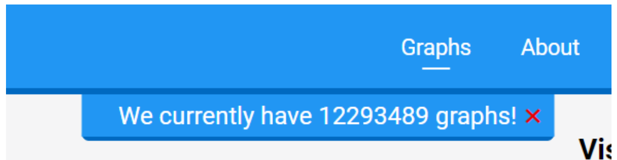
Na stejném principu probíhají i ostatní funkce komunikující s backendem.

Kapitola 5
Uživatelská příručka

V této sekci se podíváme na to, co je k vidění ve výsledné aplikaci, jak s ní pracovat a jak ji například obohatit o nové vlastnosti grafů nebo formáty stahování.

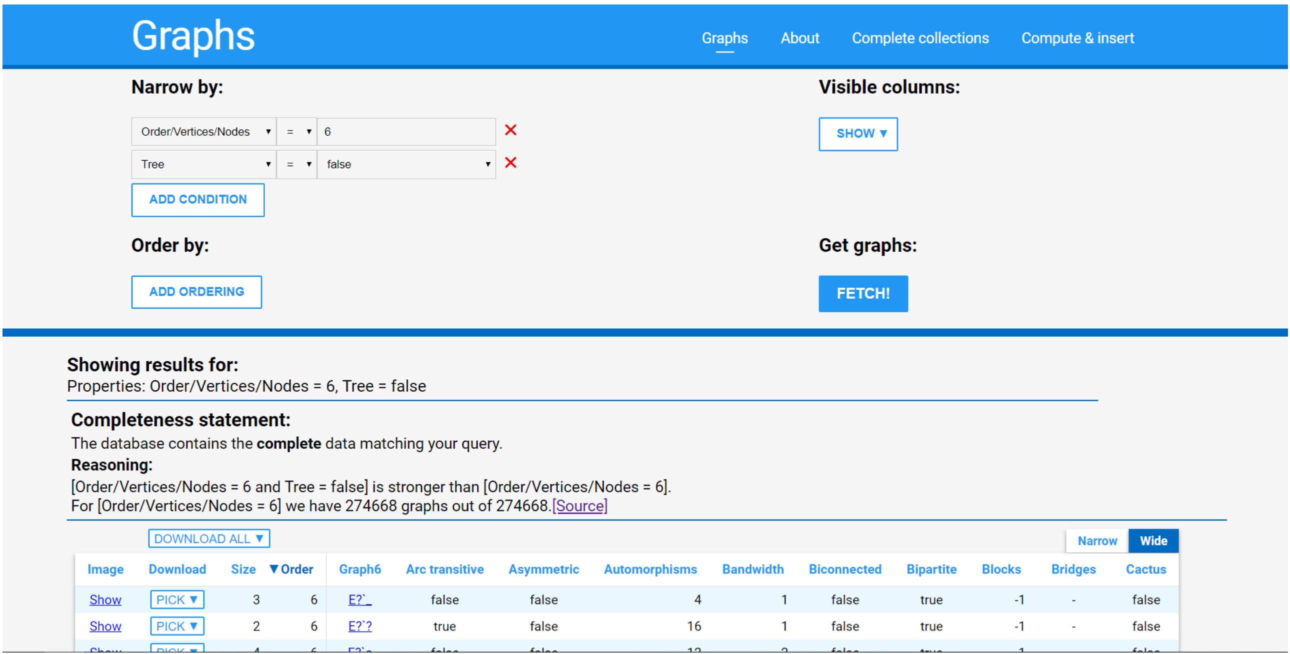
5.1 Webová aplikace

V době tvoření této práce má aplikace celkem 5 stránek, které jsou popsané níže. Aplikace je k nalezení na `http://graphs.felk.cvut.cz/`. Ve chvíli příchodu do aplikace vyskočí pro zajímavost notifikace se současným počtem grafů v databázi.



Obrázek 5.1. Notifikace počtu grafů v databázi

5.1.1 Graphs



Obrázek 5.2. Úvodní stránka, kde lze procházet grafy. [37]

Zde lze procházet všechny grafy existující v databázi. Tento proces usnadňuje filtrování (`Narrow by:`), kde si uživatel může omezit zobrazování výsledků podle určitých hodnot vlastností grafů. Pokud ho zajímají pouze grafy s 5 a méně vrcholy, omezením dotazu zúží hledaný seznam.

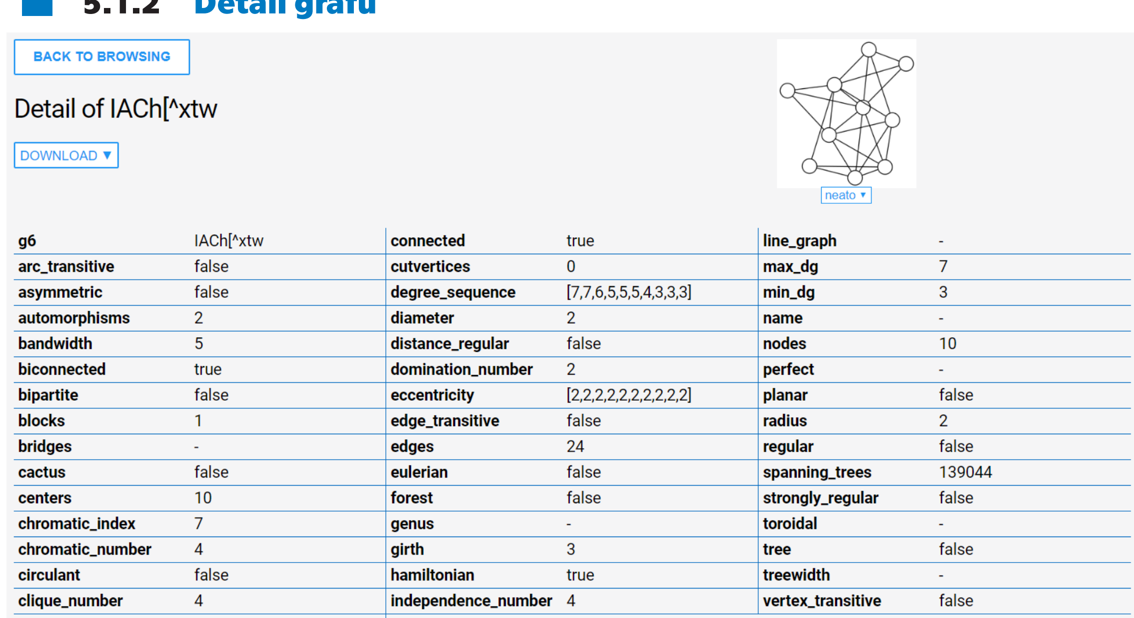
Hned pod podmínkami je možnost určení pořadí (`Order by:`), ve kterém se data vrátí z databáze. Používá se v případě, že chceme například vidět všechny grafy vzestupně podle počtu hran.

Ve `Visible columns:` je možnost si v reálném čase vybírat viditelné sloupečky v tabulce. V době psaní této práce je k dispozici 46 vlastností, což je na běžných monitorech nepřehledné, a tak je možnost se dívat pouze na ty zajímavější vlastnosti. Tlačítko `FETCH!` pošle dotaz na server. Typická doba odezvy je kolem 100ms a vždy se vrací maximálně 100 grafů. Po úspěšném návratu výsledků se pod dělící čarou objeví informace o zadaných podmínkách a o kompletnosti dotazu, zda jsou v databázi pro daný dotaz všechny grafy, nebo ne. Dále je k vidění tabulka s výsledky. V ní je možné si zvolit kompaktní verzi kliknutím na `Narrow`. K dispozici je hromadné stahování grafů tlačítkem `DOWNLOAD ALL` i jednotlivé ve sloupečku `Download`. Klikáním na nadpisy jednotlivých sloupečků je možnost řadit aktuální výsledky podle zvolené priority. Kliknutím na `Show` nebo `Graph6` kód se uživatel dostane na detail grafu. Úplně dole se nachází stránkování.

Obrázek 5.3. Stránkování

Jelikož dotaz je omezený na 100 grafů, ale podmínky jich může splňovat několik milionů, tak stránkování zaručí, že člověk není omezený na prvních 100, ale v případě zájmu se může podívat i na další, podle řazení méně prioritní grafy.

5.1.2 Detail grafu



g6	IACh[^xtw	connected	true	line_graph	-
arc_transitive	false	cutvertices	0	max_dg	7
asymmetric	false	degree_sequence	[7,7,6,5,5,5,4,3,3,3]	min_dg	3
automorphisms	2	diameter	2	name	-
bandwidth	5	distance_regular	false	nodes	10
biconnected	true	domination_number	2	perfect	-
bipartite	false	eccentricity	[2,2,2,2,2,2,2,2,2,2]	planar	false
blocks	1	edge_transitive	false	radius	2
bridges	-	edges	24	regular	false
cactus	false	eulerian	false	spanning_trees	139044
centers	10	forest	false	strongly_regular	false
chromatic_index	7	genus	-	toroidal	-
chromatic_number	4	girth	3	tree	false
circulant	false	hamiltonian	true	treewidth	-
clique_number	4	independence_number	4	vertex_transitive	false

Obrázek 5.4. Detail grafu IACh[^xtw [38]

Na detailu grafu jsou přehledně vidět všechny vlastnosti, je znovu možnost si ho stáhnout a k nahlédnutí je i obrázek s přepínáním generovacího enginu pro různé interpretace daného grafu.

5.1.3 About

Stránka s informacemi o celém projektu, jeho motivech a budoucnosti. Také zde jsou odkazy na zdroje grafů a dat, relevantní software a další příbuzné stránky.

5.1.4 Complete collections

Complete collections

Listed below, you can see the groups of graphs that are **complete** in our database, i. e. there exists **no graph** that has the given properties and is not included by us.

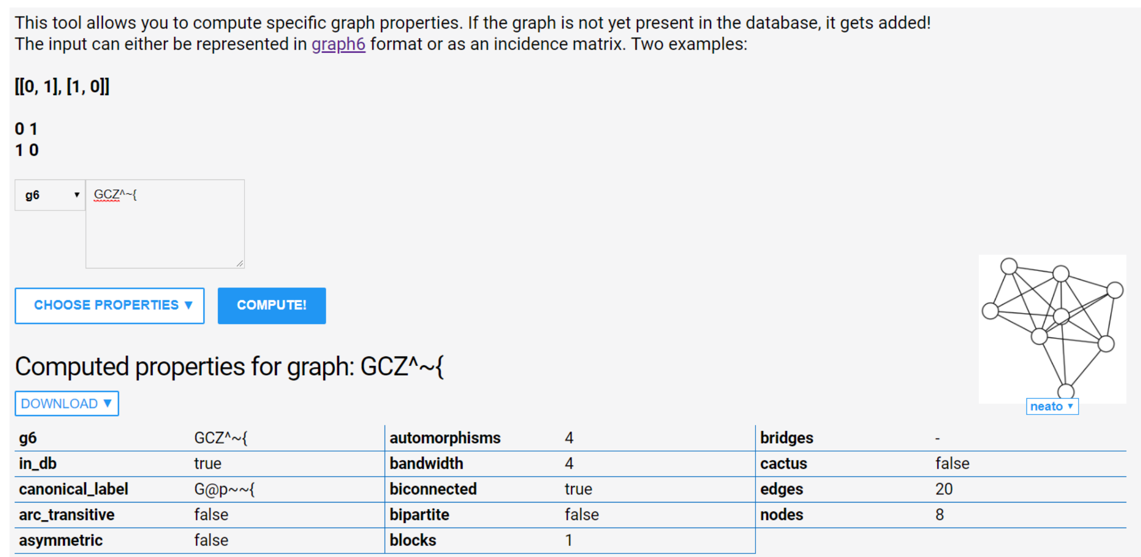
*There **may** be some sets of graphs we are sure to include at their **entirety**, but **we couldn't compute** their significant attributes yet. For that case we provide the number of*

Properties	Info
Order/Vertices/Nodes = **10**	Computed on **12005168** graphs out of **12005168** possible Source(s): OEIS: A000088
Order/Vertices/Nodes = **9**	Computed on **274668** graphs out of **274668** possible Source(s): OEIS: A000088
Order/Vertices/Nodes = **8**	Computed on **12346** graphs out of **12346** possible Source(s): OEIS: A000088
Order/Vertices/Nodes = **7**	Computed on **1044** graphs out of **1044** possible Source(s): OEIS: A000088
Order/Vertices/Nodes = **6**	Computed on **156** graphs out of **156** possible Source(s): OEIS: A000088
Order/Vertices/Nodes = **5**	Computed on **34** graphs out of **34** possible Source(s): OEIS: A000088
Order/Vertices/Nodes = **4**	Computed on **11** graphs out of **11** possible Source(s): OEIS: A000088
Order/Vertices/Nodes = **3**	Computed on **4** graphs out of **4** possible Source(s): OEIS: A000088
Order/Vertices/Nodes = **2**	Computed on **2** graphs out of **2** possible Source(s): OEIS: A000088
Order/Vertices/Nodes = **1**	Computed on **1** graphs out of **1** possible Source(s): OEIS: A000088

Obrázek 5.5. Stránka o kompletnosti grafů [39]

Stránka s informacemi o kompletnosti databáze. Informuje uživatele o tom, které druhy grafů máme v databázi shromážděné všechny a které ne. Backend k této funkcionalitě byl vyvíjen paralelně s touto aplikací[40].

5.1.5 Compute & insert

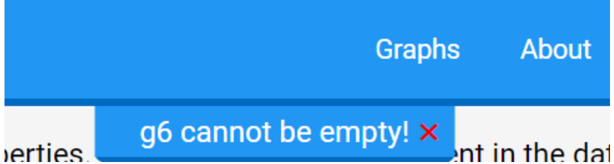


g6	GCZ^~{	automorphisms	4	bridges	-
in_db	true	bandwidth	4	cactus	false
canonical_label	G@p~~{	biconnected	true	edges	20
arc_transitive	false	bipartite	false	nodes	8
asymmetric	false	blocks	1		

Obrázek 5.6. Spočtené vlastnosti grafu GCZ^~{ [41]

Zde se počítají vlastnosti grafů přítomných i nepřítomných v databázi. Pokud graf ještě v databázi neexistuje, je přidán. Uživatel svůj graf může zadávat ve formátu g6 i jako matici sousednosti. Vybere si požadované vlastnosti a kliknutím na tlačítko COMPUTE!

si zažádá o výsledek. V případě validního grafu se mu zobrazí výsledné vlastnosti i obrázek. Formáty musí být dodrženy přesně tak, jak je uvedeno na stránce. V případě, že se tak nestane a nebo vstup není validní v jiném smyslu, třeba překlep v `g6`, se uživateli objevuje notifikace, která vypadá následovně.



Obrázek 5.7. Notifikace chyby

5.2 Přidávání vlastností grafů

V nevyhnutelné situaci objevení nebo vymyšlení nových vlastností grafů je žádané, aby jejich přidání do rozhraní nebylo nijak náročné. Vlastnost stačí přidat jako sloupeček databáze a potom v souboru `src/config.js` upravit proměnnou `graphColumnsArray`, která je polem všech dosavadních vlastností.

5.2.1 config.js

```
{
  name: 'edges',
  tableDisplayName: 'Size',
  displayName: 'Size/Edges',
  type: 'number',
  visibleByDefault: true,
  fixed: true,
  computable: true
}
```

Ukázka jedné vlastnosti grafu v konfiguračním souboru. Na konec pole stačí zkopírovat řádek s jinou vlastností, přepsat její název, text pro zobrazení v tabulce a typ. Další dostupné vlastnosti jsou, zda je vlastnost implicitně vidět ve výsledcích. Protože vlastností je opravdu hodně, je preferované nejdříve zobrazovat jen ty důležitější a zbytek už si může naklikat uživatel v rozhraní. Potom můžeme nastavit, zda je vlastnost velmi důležitá a bude ve fixní části tabulky a nakonec jestli je spočítatelná v kalkulačce.

5.3 Přidávání formátů stahování

V době vypracování této bakalářské práce jsou k dispozici tři formáty pro stažení jednoho grafu a dva formáty pro stahování hromadné.

- V `json` formátu lze stáhnout objekt jednoho grafu i pole objektů více grafů.
- Ve formátu `csv` lze stahovat jeden i více grafů. Každý graf je na jednom řádku.
- `Matici sousednosti` si lze stáhnout pouze pro jeden graf.

5.3.1 Download.js

V souboru `src/graphs/components/Download.js` je možné najít `options`, kde jsou definovány dostupné formáty. Standardní `json` a `csv` je dostupný pro více grafů najednou, ale matice je stažitelná jen pro jednotlivé grafy. V případě, že někdy bude žádané přidat nový formát, dá se na něj napsat `JavaScript` funkce například do souboru `src/graphs/lib/helpers`, poté naimportovat v `Download.js`, podobně jak se to již děje níže a podle vzoru připsat nový formát.

```
import { JSONToCSV } from '../lib/helpers'
import G6 from '../../../utils/g6'
.
.
.
const { graph } = this.props
const options = [
      {
        format: 'json',
        filename: 'graph.json',
        download: `data:application/json;charset=utf-8,
        ${encodeURIComponent(JSON.stringify(graph))}`,
      },
      {
        format: 'csv',
        filename: 'graph.csv',
        download: `data:text/csv;charset=utf-8,
        ${encodeURIComponent(JSONToCSV(graph))}`,
      },
    ]
    if (!Array.isArray(graph)) {
      const matrix = new G6(graph.g6).toMatrix()
      const matrixCSV = matrix.join('\r\n')
      options.push({
        format: 'matrix',
        filename: 'graph.csv',
        download: `data:text/csv;charset=utf-8,
        ${encodeURIComponent(matrixCSV)}`,
      })
    }
```

Ukázka kódu 5.1. Ukázka kódu ze souboru `src/graphs/components/Download.js`

Kapitola 6
Další rozšiřování a nedostatky

Vývojáři mají úplnou svobodu přidávat a rozšiřovat komponenty i styly. V aplikaci není nic schovaného ani natvrdo přimontovaného. Změny jsou otázkou přepisování již existujícího kódu a nebo tvořením a používáním nových komponent.

6.1 Další rozšiřování

Budoucnost aplikace do sebe rozhodně zahrnuje přidávání vlastností a formátů, jak bylo zmíněno v uživatelské příručce, ale i nedostatky sepsané níže. Dále mezi možná rozšíření patří i implementace podpory většiny rozšířených grafových datých formátů jako: `Sparse6, GraphML, DXL, LEDA` apod. Zajímavá by byla i integrace některého z hotových grafových editorů.

6.2 Nedostatky

Aplikace splňuje očekávané funkce, ale z hlediska webového vývoje jsme si vědomi nedokonalé mobilní responsivity. U obrovských tabulek dat je obtížné vymyslet jejich vhodnou reprezentaci na úzkých zařízeních. V současnou chvíli a vzhledem k typu potenciálních uživatelů aplikace však bohatě stačí optimalizovaný uživatelský zážitek pro stolní počítače a notebooky. Mobilní responsivita tabulky výsledků tedy byla v současné verzi považována za nad rámcovou, ale může být implementována v dalších verzích.
Neoptimální může být i chování v prohlížeči `Internet Explorer` starším než verze 9. Zastoupení uživatelů je tak malé, že se na něj vzhledem k současným potřebám nebral ohled.

Kapitola 7
Závěr

V této práci jsme přinesli řešení pro práci s nepřeberným množstvím grafů. Byla vytvořena aplikace, která umí komunikovat s paralelně vyvíjeným backendem a umožňuje získávat výsledky z databáze podle požadovaných kritérií. Těmito kritérii jsou pořadí výsledků podle dané vlastnosti a nebo specifické vlastnosti omezené podmínkami jako například `'Všechny grafy s pěti a méně vrcholy'`. Tato kritéria se dají různě kombinovat a sčítat. Tabulka s výsledky je přehledná a pracuje se s ní intuitivně a jednoduše. Shrnuté vlastnosti grafu si uživatel může prohlédnout na detailu grafu, společně s jeho obrázkem vygenerovaného v různých enginech. Grafy je možno stahovat například ve formátech `json` a `csv`.
Implementovala se i stránka, kde se dá interagovat s kalkulátorem vlastností, který přijímá grafy ve formátu `g6` a grafy jako matici sousednosti. Grafy, které ještě nejsou přítomné v databázi, se do ní prostřednictvím kalkulátoru dají přidávat.
Celý vývojový proces je usnadněný díky `Node.js` a `Webpacku`, takže v případě budoucí práce dalších programátorů už je připravené příjemné prostředí.
Aplikace má potenciál na rozšiřování popsaný v předešlé kapitole, ale už v současném stavu je připravena na reálné a přínosné využívání.

Literatura

[1] *HTML wiki* [
https://en.wikipedia.org/wiki/HTML].

[2] *CSS wiki* [
https://en.wikipedia.org/wiki/Cascading_Style_Sheets].

[3] *JavaScript wiki* [
https://en.wikipedia.org/wiki/JavaScript].

[4] Crockford, Douglas. *JavaScript: The Good Parts*. Yahoo Press, 2008.

[5] *DOM wiki* [
https://en.wikipedia.org/wiki/Document_Object_Model].

[6] *HTML5 intro* [
https://www.w3schools.com/html/html5_intro.asp].

[7] *Internet live stats* [
http://www.internetlivestats.com/total-number-of-websites].

[8] *Caniuse grid layout* [
https://caniuse.com/#feat=css-grid].

[9] *Caniuse homepage* [
https://caniuse.com/].

[10] *Bootstrap homepage* [
https://getbootstrap.com/].

[11] *Semantic UI homepage* [
https://semantic-ui.com/].

[12] *Preprocessors* [
https://learn.shayhowe.com/advanced-html-css/preprocessors/#scss-sass].

[13] *Sass homepage* [
https://sass-lang.com/].

[14] *Node.js homepage* [
https://nodejs.org/en/].

[15] *JavaScript versions* [
https://www.w3schools.com/js/js_versions.asp].

[16] *Babel homepage* [
https://babeljs.io/].

[17] *Electron homepage* [
https://electronjs.org/].

[18] *React Native* [
https://facebook.github.io/react-native/].

[19] *NPM homepage* [
https://www.npmjs.com/].

[20] *Module counts* [
http://www.modulecounts.com/].

[21] *React homepage* [
https://reactjs.org/].

[22] *React JSX* [
https://reactjs.org/docs/introducing-jsx.html].

[23] *React lifecycle methods* [
https://reactjs.org/docs/state-and-lifecycle.html].

[24] *Redux homepage* [
https://redux.js.org/].

[25] *React Router homepage* [
https://reacttraining.com/react-router/core/guides/philosophy].

[26] *Webpack homepage* [
https://webpack.js.org/].

[27] *Git homepage* [
https://git-scm.com/].

[28] Roun, Tomáš. *Graph Database Fundamental Services*. 2018.

[29] *Express.js homepage* [
https://expressjs.com/].

[30] *Fetch API doc page* [
https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API].

[31] *Google Chrome homepage* [
https://www.google.com/chrome/].

[32] *Firefox homepage* [
https://www.mozilla.org/en-US/firefox/new/].

[33] *Internet Explorer 11 wiki* [
https://en.wikipedia.org/wiki/Internet_Explorer_11].

[34] *Polyfill wiki* [
https://en.wikipedia.org/wiki/Polyfill_(programming)].

[35] *Fetch doc page* [
https://developers.google.com/web/updates/2015/03/introduction-to-fetch].

[36] *Promises doc page* [
https://www.promisejs.org/].

[37] *Graphs homepage* [
http://graphs.felk.cvut.cz/].

[38] *Graphs detail page* [
http://graphs.felk.cvut.cz/detail/IACh[%5Extw].

[39] *Graphs completeness page* [
http://graphs.felk.cvut.cz/complete].

[40] Ullrich, Herbert. *User Extensible Graph Database*. 2018.

[41] *Graphs computer page* [
http://graphs.felk.cvut.cz/computer].