Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Vladimir Bajt

Vizualizacija podatkov v JavaScriptu

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Tomaž Dobravec

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Javascript je programski jezik, ki omogoča izvajanje programov na strani odjemalca. Jezik podpira večina spletnih brskalnikov, zato njegova uporaba omogoča zanesljivo in nemoteno delovanje spletnih strani.

V diplomskem delu preglejte možnost uporabe programskega jezika Javascript za vizualizacijo podatkov v brskalniku. Predstavite spletne tehnologije in standarde (HTML, DOM, JSON, Ajax, ...) ter orodja (Flask, Bootstrap, jQuery, ...), ki se pogosto uporabljajo pri izdelavi javascript spletnih aplikacij. Na osnovi knjižnic C3.js in D3.js izdelajte spletno aplikacijo za izdelavo poizvedb v sistemu ALGator in za prikaz rezultatov teh poizvedb v grafični obliki. Opišite tudi primere uporabe izdelane aplikacije.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Vladimir Bajt, z vpisno številko **63050006**, sem avtor diplomskega dela z naslovom:

Vizualizacija podatkov v JavaScriptu

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Tomaža Dobravca,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 22. septembra 2014 Podpis avtorja:

Zahvaljujem se mentorju doc. dr. Tomažu Dobravcu za strokovno pomoč in svetovanje pri pisanju diplomskega dela.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
HTTP	HyperText Transfer Protocol	protokol za izmenjavo nadbesedila
HTML	HyperText Markup Language	jezik za označevanje nadbesedila
DOM	Document Object Model	objektni model dokumenta
SVG	Scalable Vector Graphics	umerljiva vektorska grafika
CSS	Cascading Style Sheets	kaskadna slogovna predloga
AJAX	Asynchronous JavaScript and XML	asinhroni JavaScript in XML
JSON	JavaScript Object Notation	JavaScript objektna notacija
URL	Uniform Resource Locator	enolični krajevnik vira
API	Application programming interface	programski vmesnik

Povzetek

Diplomsko delo obravnava temo vizualizacije podatkov v spletnih aplikacijah z uporabo programskega jezika JavaScript. Glavni cilj vizualizacije podatkov je grafična predstavitev podatkov na način, ki uporabniku omogoča učinkovito razumevanje posredovane informacije. Delo obsega predstavitev navadno uporabljenih tehnologij za razvoj modernih spletnih aplikacij, predvsem smo si podrobneje ogledali delovanje tehnologije SVG (ang. Scalable Vector Graphics) in programsko knjižnico D3.js. Z uporabo predstavljenih tehnologij in orodij smo razvili spletno aplikacijo za grafični in tabelarični prikaz podatkov. Aplikacijo sestavljata dve glavni komponenti, komponenta za sestavljanje poizvedb in komponenta za risanje različnih vrst grafov. Izmenjava podatkov med spletno aplikacijo in strežnikom poteka asinhrono v ozadju z uporabo tehnike poimenovane AJAX. Komponente aplikacije smo zasnovali modularno, kar pomeni, da se lahko posamezne komponente uporabi samostojno, neodvisno od ostalih modulov.

Ključne besede: D3.js, jQuery, vizualizacija podatkov, JavaScript, SVG.

Abstract

This thesis presents the topic of data visualization in web applications using the programming language JavaScript. Main goal of data visualization is to communicate information to the user effectively and clearly through graphical means. Thesis includes an overview of technologies and tools commonly used for developing modern web-based applications. In particular, we take a closer look at SVG (Scalable Vector Graphics) and D3.js, a JavaScript library for manipulating documents. Using the presented technologies and tools we developed a web-based application for graphic and tabular display of data. The application consists of two main components, a query editor, which provides an interface for creating queries, and a graph component that is used to draw different types of graphs. Data between the web application and the server is exchanged asynchronously using AJAX technique. Application components are designed to be modular, meaning that individual components can be used independently.

Keywords: D3.js, jQuery, data visualization, JavaScript, SVG.

Poglavje 1

Uvod

Spletne aplikacije so v zadnjih nekaj letih doživele korenite spremembe. Iz zbirke preprostih in statičnih spletnih strani so zrasle v interaktivne in dinamične aplikacije z bogatimi uporabniškimi vmesniki, ki izgledajo in delujejo, kot tisti v tradicionalnih namiznih aplikacijah. V preteklosti [1] je interaktivnost v spletnih aplikacijah bila dosežena predvsem z uporabo orodij kot je Flash. Starejši JavaScript pogoni so bili neučinkoviti in neprimerni za izvajanje kompleksnih operacij, prikaz vektorske grafike in animacij. V spletnih aplikacijah je bila uporaba JavaScripta večinoma omejena na validacijo obrazcev in druge preproste operacije. Danes so sodobni brskalniki precej bolj učinkoviti in se zato JavaScript v vedno večji meri uporablja za razvoj interaktivnih spletnih aplikacij.

Diplomsko delo obravnava temo vizualizacije podatkov v spletnih aplikacijah z uporabo programskega jezika JavaScript. Ogledali smo si delovanje tehnologij in programskih knjižic, ki se danes uporabljajo za razvoj modernih spletnih aplikacij. V ospredje smo postavili predvsem delovanje tehnologije SVG (ang. Scalable Vector Graphics) in programske knjižnice D3.js za namen grafičnega prikaza podatkov. Kot praktični del diplomske naloge, smo razvili spletno aplikacijo, ki nam omogoča vizualizacijo in enostavno primerjavo rezultatov med različnimi algoritmi pri reševanju nekega problema.

V uvodnem poglavju je na kratko predstavljeno diplomsko delo in njegova zgradba. V poglavjih dve in tri sledi predstavitev spletnih tehnologij in programskih knjižnic ter orodij uporabljenih za izdelavo diplomskega dela. Jedro dela predstavlja opis izdelave spletne aplikacije kjer je podrobneje opisan namen, struk-

tura in prikaz uporabe aplikacije. V poglavju pet je predstavljena konfiguracija in namestitve razvite rešitve. Na koncu se nahaja še zaključek.

Poglavje 2

Spletne tehnologije

2.1 HTML

HTML (ang. HyperText Markup Language) je označevalni jezik namenjen izdelavi spletnih aplikacij. Vsebino HTML dokumenta označimo s pomočjo značk, katere uporabimo za določanje semantične strukture našega dokumenta. Značke se lahko pojavljajo v parih, v takem primeru se naša vsebina nahaja med začetno in končno značko, npr. značka za odstavke `<p>`. Preprost HTML dokument je videti tako:

```
<!DOCTYPE html>
<html>
        <head>
                <title>Naslov strani</title>
        </head>
        <body>
                <p>Prvi odstavek</p>
        </body>
</html>
```

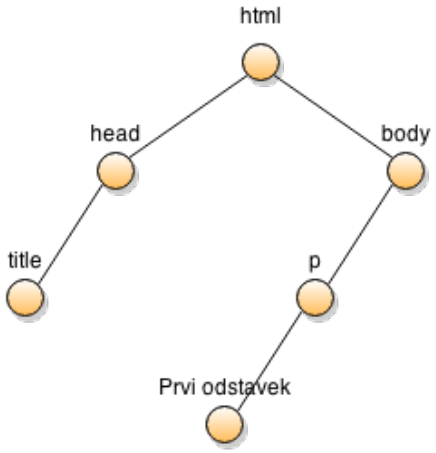
HTML5 [16] je naslednja generacija HTML jezika, ki nadomešča stare verzije. Prinaša novosti in spremembe na večih področjih:

- nove značke namenjene boljši semantični podpori dokumentov (`<article>`, `<nav>`, `<header>`, `<footer>`, ...),
- programski vmesnik za delovanje aplikacij brez povezave do interneta,

- programski vmesnik za predvajanje avdio in video vsebin (`<audio>`, `<video>`),
- programski vmesnik za manipulacijo zgodovine brskalnika s pomočjo `history` objekta. V naši aplikaciji uporabljamo metodo `history.replaceState()` za nadomestitev trenutnega vnosa zgodovine,
- programski vmesnik za "povleci in spusti" (ang. Drag and Drop API).

2.2 DOM

DOM [11] (ang. Document Object Model) je programski vmesnik, ki zagotavlja strukturirano predstavitev dokumenta in opredeljuje način dostopa do elementov dokumenta. Strukturo HTML dokumenta si lahko predstavljamo kot drevo, vsaka značka (oziroma par značk) je element v drevesu. V samem korenu drevesa se nahaja html element, vsi ostali elementi so potomci. DOM drevo je prikazano na sliki 2.1.



Slika 2.1: Preprosto DOM drevo.

2.3 CSS

CSS (ang. Cascading Style Sheets) se uporablja za nastavljanje oblikovnih lastnosti elementov. CSS stili so sestavljeni iz selektorjev in oblikovnih lastnosti. Selektorji

opredeljujejo specifične elemente, za katere bo uporabljen določen stil. Struktura preprostega CSS dokumenta je videti tako:

```
selektor {
        lastnostA: vrednost;
        lastnostB: vrednost;
}
```

2.4 JavaScript

JavaScript [18] je objektno usmerjeni skriptni jezik. Navadno je uporabljen v spletnih aplikacijah, v zadnjih letih pa tudi v strežniških okoljih. V brskalnikih se JavaScript v navezi z DOM programskim vmesnikom uporablja za upravljanje vsebine in strukture HTML dokumentov.

2.5 JSON

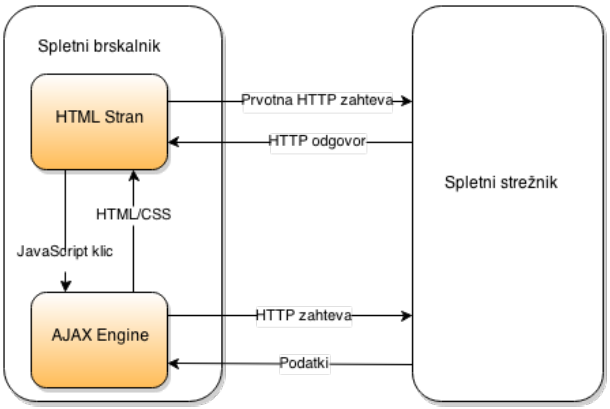
JSON (ang. JavaScript Object Notation) je format za izmenjavo podatkov v človeku razumljivi obliki. Spominja na podmnožico programskega jezika JavaScript, vendar je neodvisen od JavaScripta. JSON zapis podpira osnovne podatkovne tipe: število, niz, seznam, objekt, logični vrednosti `true` in `false` ter `null`. Primer JSON zapisa:

```
{
        "id": 1,
        "ime": "imeA",
        "cena": 9.9,
        "lastnosti": ["lastnostA", "lastnostB"]
}
```

2.6 AJAX

AJAX [6] (ang. Asynchronous JavaScript + XML) ni tehnologija sama po sebi, ampak je izraz skovan leta 2005, ki opisuje novi pristop k skupni uporabi obstoječih

tehnologij, kot so HTML in XHTML, CSS, JavaScript, DOM, XML in XMLHttpRequest. Spletne aplikacije so s pomočjo AJAX modela zmožne posodobitve dela dokumenta brez ponovnega nalaganja celotne strani. Na sliki 2.2 je prikazan AJAX komunikacijski model.



Slika 2.2: AJAX komunikacijski model.

Podatke lahko prejemamo in pošiljamo v različnih zapisih, vključno z XML, HTML, JSON in tekstom. V JavaScriptu HTTP zahtevo ustvarimo s pomočjo `XMLHttpRequest` objekta:

```
//ustvarimo novo http zahtevo
var request = new XMLHttpRequest();

//funkcija, ki se klice ko prejmemo odgovor
request.onreadystatechange = handlerFunc;

//posljimo zahtevo
request.open("GET", url);
request.send();
```

2.7 Python

Python je dinamični visokonivojski programski jezik, s pomočjo katerega lahko hitro razvijamo aplikacije. Njegova zasnova poudarja berljivost in izrazitost programske kode. Na voljo je na večih platformah, vključno z Unix/Linux, Mac OSX

in Windows. V naši aplikaciji uporabljamo Python skupaj z ogrodjem za izdelavo spletnih aplikacij Flask [15].

2.8 SVG

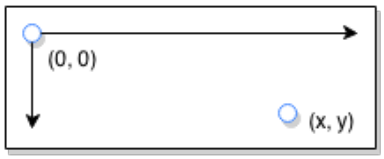
SVG [23] (ang. Scalable Vector Graphics) je označevalni jezik namenjen opisovanju dvodimenzionalne vektorske grafike. SVG kodo lahko vključimo neposredno v katerikoli HTML dokument, lahko jo pa vključimo tudi dinamično v sam DOM dokumenta. SVG podpirajo vsi moderni spletni brskalniki z izjemo brskalnika Internet Explorer verzije 8 in starejše verzije.

SVG element ustvarimo s pomočjo značke `<svg>`. Elementu določimo višino in širino, v nasprotnem primeru bo zavzel ves prostor, ki je na voljo znotraj nadrejenega elementa.

```
<svg width="250" height="250">
</svg>
```

2.8.1 Koordinatni sistem

SVG koordinatni sistem temelji na slikovnih pikah (ang. pixels), kjer se izhodišče koordinatnega sistema (0,0) nahaja v zgornjem levem kotu, kot je prikazano na sliki 2.3. S povečevanjem x vrednosti se pomikamo proti desni, medtem ko se s povečevanjem y vrednosti pomikamo navzdol.



Slika 2.3: SVG koordinatni sistem.

2.8.2 Element `<rect>`

Element `<rect>` se uporablja za risanje pravokotnikov. Atributa `x` in `y` določata kje se nahaja zgornji levi kot pravokotnika, medtem ko `width` in `height` določata širino

in višino pravokotnika. Atribut `fill` določa barvo v notranjosti pravokotnika, `stroke` pa določa barvo obrobe:

```
<rect fill="red" stroke="black" x="0" y="0" width="200" height="300" />
```

2.8.3 Elementa <circle> in <ellipse>

Element `<circle>` nariše krog. Atributa `cx` in `cy` določata kje se nahaja središče, medtem ko atribut `r` določa polmer kroga. Podobno velja za `<ellipse>` z razliko, da določimo vrednosti polmera za posamezno os. Oba elementa sta prikazana na sliki 2.4.

```
<circle fill="green" cx="100" cy="50" r="45" />
<ellipse fill="red" cx="250" cy="50" rx="75" ry="50" />
```

Slika 2.4: Prikaz `<circle>` in `<ellipse>`.

2.8.4 Element <line>

Risanje črt je mogoče z uporabo `<line>` značke. Atributa `x1` in `y1` določata začetne koordinate črte, medtem ko `x2` in `y2` določata konec.

```
<line x1="50" y1="50" x2="250" y2="250" stroke="red" />
```

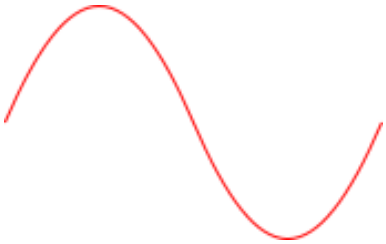
2.8.5 Element <path>

Element `<path>` [22] se uporablja za risanje zapletenih poti. Obliko poti opredeljuje atribut `d`.

```
<path d="M 10 80 C 40 10, 65 10, 95 80 S 150 150, 180 80"
        stroke="red" fill="white" />
```

Atribut `d` vsebuje niz ukazov in njihovih parametrov, ki določajo končno obliko poti. Na voljo je več vrst ukazov. Vsak ukaz se začne z določeno črko. Na primer ukaz `moveto` se začne s črko M. V primeru, da ukaz začnemo z veliko črko (M, L, C, ...), so podane vrednosti koordinat absolutne. V primeru, da ukaz začnemo z malo črko (m, l, c, ...), pa so podane vrednosti koordinat relativne. Prikaz elementa se nahaja na sliki 2.5.

- `moveto` (M) - začetek nove poti ali podpoti,
- `closepath` (Z) - konec trenutne poti ali podpoti,
- `lineto` (L) - nariše ravno črto od trenutne točke do nove točke,
- `curveto` (C) - nariše krivuljo od trenutne točke do nove točke.



Slika 2.5: Prikaz `<path>`.

2.8.6 Element `<text>`

Element `<text>` opredeljuje grafični element, sestavljen iz besedila. Mogoča je uporaba oblikovnih atributov `style`, `font-family` in `font-size`.

```
<text x="50" y="50">SVG tekst</text>
```

2.8.7 Element `<polygon>`

Opredeljuje zaprto grafično obliko, sestavljeno iz ravnih črt, kot vidimo na sliki 2.6. Atribut `points` vsebuje niz vrednosti (x,y), ki določajo robove oblike.

```
<polygon points="20,20 100,20 100,80 20,100" fill="pink" />
```

Slika 2.6: Prikaz `<polygon>`.

2.8.8 Element `<polyline>`

Opredeljuje odprto grafično obliko, sestavljeno iz ravnih črt, ki povezujejo več točk (slika 2.7).

```
<polyline stroke="blue" fill="none"
        points="10,100 50,70 100,90 150,10 200,90 250,70 300,100" />
```

Slika 2.7: Prikaz `<polyline>`.

2.8.9 Element `<g>`

Element `<g>` (slika 2.8) opredeljuje skupino predmetov. Transformacije in atribute uporabljene na skupini podedujejo njegovi otroci:

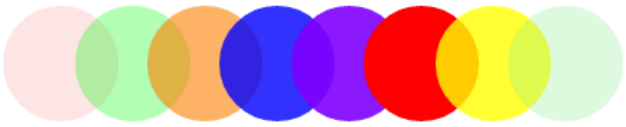
```
<g fill="steelblue" stroke="rgba(0, 255, 0, 0.3)" stroke-width="10">
   <rect x="50" y="50" width="80" height="80" />
   <rect x="150" y="50" width="80" height="80" />
   <rect x="250" y="50" width="80" height="80" />
</g>
```

Slika 2.8: Prikaz skupine.

2.8.10 Plasti in transparentnost

SVG ne pozna plasti in koncepta globine. Prav tako ne podpira CSS lastnosti `z-index`, oblike lahko razvrščamo samo v dvodimenzionalni ravnini. Elementi, ki so v dokumentu narisani pozneje, prekrijejo elemente narisane na isti poziciji. Zadnji element v dokumentu je narisan na vrhu vsega drugega. Pri poudarjanju elementov je zelo uporabna transparentnost. Na voljo sta nam dva načina nastavljanja transparentnosti, z uporabo atributa `opacity` in nastavljanje `rgba()` vrednosti. Nastavljanje transparentnosti je vidno na sliki 2.9.

```
<circle cx="50" cy="45" r="40" fill="rgba(255, 0, 0, 0.1)" />
<circle cx="100" cy="45" r="40" fill="rgba(0, 255, 0, 0.3)" />
<circle cx="150" cy="45" r="40" fill="rgba(255, 128, 0, 0.6)" />
<circle cx="200" cy="45" r="40" fill="rgba(0, 0, 255, 0.8)" />
<circle cx="250" cy="45" r="40" fill="rgba(128, 0, 255, 0.9) "/>
<circle cx="300" cy="45" r="40" fill="rgba(255, 0, 0, 1.0)" />
<circle cx="350" cy="45" r="40" fill="yellow" opacity="0.8" />
<circle cx="400" cy="45" r="40" fill="lightgreen" opacity=0.3 />
```



Slika 2.9: Nastavljanje transparentnosti.

Poglavje 3

Programske knjižnice in orodja

3.1 Flask

Flask [3] je odprtokodno spletno ogrodje za razvoj aplikacij. Preprosta aplikacija je prikazana na sliki 3.1. Flask temelji na programskem jeziku Python. Ogrodje vsebuje:

- razvojni strežnik in razhroščevalnik za hiter razvoj aplikacij,
- Integrirano podporo za testiranje enot (ang. unit testing),
- Jinja2 predloge,
- podporo za Unicode,
- podporo za seje.

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def index():
    return "Preprosta Flask aplikacija"

if __name__ == "__main__":
    app.run()
```

Slika 3.1: Preprosta Flask aplikacija

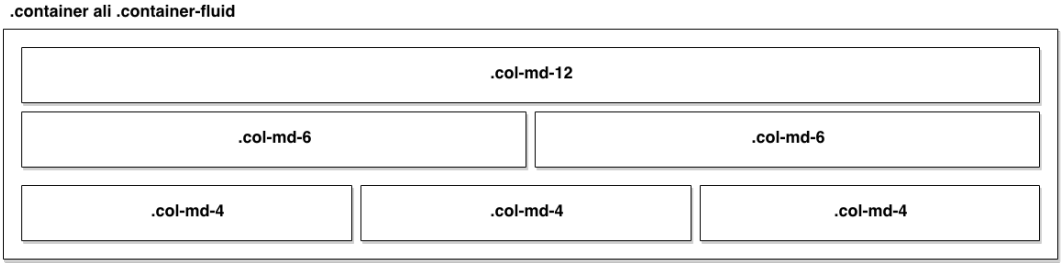
3.2 Bootstrap

Bootstrap [7] je odprtokodna zbirka orodij za ustvarjanje spletnih strani in aplikacij. Vsebuje HTML in CSS predloge za tipografijo, obrazce, gumbe, navigacijo in ostale komponente grafičnega vmesnika. Bootstrap je združljiv z vsemi različicami modernih brskalnikih, v primeru uporabe starejših brskalnikov, kot je na primer Internet Explorer 8, elementi elegantno degradirajo. Bootstrap podpira odzivno spletno oblikovanje (ang. Responsive web design), kar pomeni, da se razporeditev spletne strani prilagodi dinamično, glede na značilnosti naprave, ki dostopa do spletne strani.

3.2.1 Mrežni sistem

Bootstrapov mrežni sistem [8] (ang. grid system) se uporablja za razporeditev (ang. layout) spletne strani. Elementi se na strani razvrstijo v vrstice in stolpce:

- Vrstice umestimo znotraj `.container` ali `.container-fluid` elementa.
- Vrstice nato uporabimo za ustvarjanje skupin stolpcev. Posamezno vrstico lahko razdelimo na največ 12 stolpcev.
- Vsebino umestimo znotraj posameznih stolpcev, samo stolpci so lahko neposredni potomci vrstic.
- Stolpcem določimo razrede, ki označujejo njihovo širino. Če želimo stolpec, ki se razteza čez šest mest uporabimo razred `.col-md-6`.

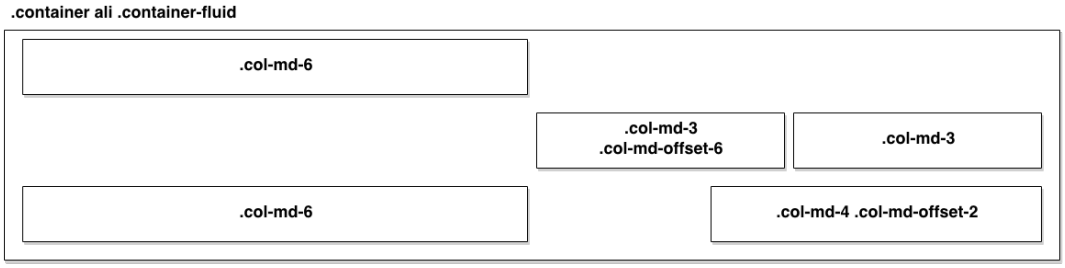


Slika 3.2: Bootstrap mrežni sistem.

Na primer, vsebino svoje strani želimo razvrstiti, kot je to storjeno na sliki 3.2. V prvi vrstici bi radi, da se vsebina razteza čez celotno širino vrstice. V drugi in tretji vrstici želimo svojo vsebino umestiti v dve oziroma tri enako široke stolpce. Naša HTML koda izgleda takole:

```
<div class="container">
        <div class="row">
                <div class="col-md-12">.col-md-12</div>
        </div>
        <div class="row">
                <div class="col-md-6">.col-md-6</div>
                <div class="col-md-6">.col-md-6</div>
        </div>
        <div class="row">
                <div class="col-md-4">.col-md-4</div>
                <div class="col-md-4">.col-md-4</div>
                <div class="col-md-4">.col-md-4</div>
        </div>
</div>
```

Zamikanje stolpcev je možno doseči z uporabo razredov `.col-md-offset-*`. Ti razredi povečajo levi rob posameznega stolpca. Če želimo zamakniti stolpec za x mest uporabimo razred `.col-md-offset-x`. Zamikanje stolpcev je prikazano na sliki 3.3.



Slika 3.3: Zamikanje stolpcev.

3.3 jQuery

jQuery [19] je JavaScript programska knjižnica, ki omogoča enostavno manipulacijo elementov v HTML dokumentih. Izdana je pod MIT licenco, izvorna koda pa je na voljo na Githubu. Knjižnica vsebuje naslednje funkcije:

- metode za izbiro in manipulacijo DOM elementov,
- metode za delo z DOM dogodki (ang. events),
- metode za delo z Ajaxom,
- metode za animacijo in efekte.

3.3.1 Funkciji `$()` in `jQuery()`

Večji del funkcionalnosti jQuery knjižnice nam je na voljo z uporabo funkcije `$()` ali malce daljše `jQuery()`. Obe imeni se nanašata na isti objekt, kar je vidno v jQuery izvorni kodi:

```
// Expose jQuery to the global object
window.jQuery = window.$ = jQuery;
```

3.3.2 Izbira elementov

Najpreprostejše opravilo, ki ga lahko opravimo z jQuery je izbira nekaj elementov v dokumentu. Funkciji `$()` podamo selektor, ki določa našo izbiro:

```
$("#text");         //izberi element z id vrednostjo "text"
$(".hidden");         //izbira elementov na podlagi razreda
$("p");         //izbira na podlagi vrste elementa
$("div");         //izbira vseh div elementov v dokumentu
$("div.nav");         //izbira vseh div elementov z razredom "nav"
$("div p");         //izbira p elementov znotraj div

//izbira checkboxov, ki se nahajajo v div elementih z razredom "container"
$("div.container input[type=checkbox]");
```

3.3.3 Ustvarjanje elementov

Funkcija `$()` se lahko uporabi tudi za ustvarjanje novih elementov:

```
$("<div>")        //ustvarimo prazen div element

//ustvarimo h1 z vsebino "Heading 1"
$("<h1>", {
        "html": "Heading 1",
        "class": "heading"
});
```

3.3.4 Delo z dogodki

Dogodek [13] predstavlja sporočilo zasnovano na `Event` vmesniku. Poznamo več vrst dogodkov, nas pa ponavadi izmed množice dogodkov zanima le majhen del. V ta namen uporabimo jQuery funkciji `on()` in `off()`, s katerima prijavimo ali odjavimo poslušalce za specifične dogodke. Poslušalec je lahko anonimna funkcija ali pa ime funkcije deklarirane drugje v programu.

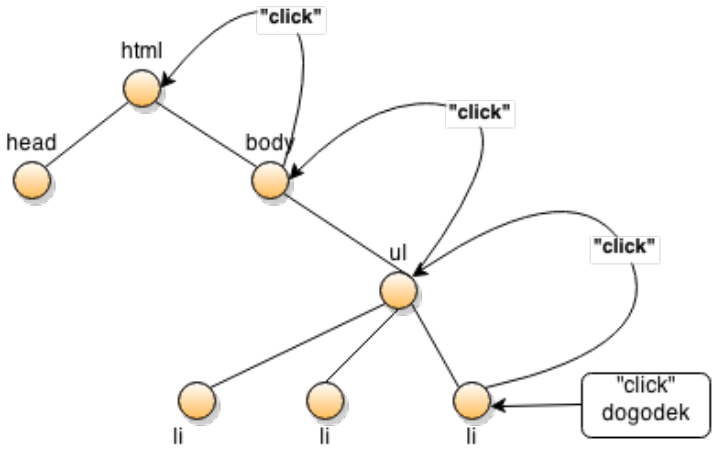
```
//<li> elementom z razredom "entry" dodamo event handler kateri
//ob kliku na posameznem <li> nastavi vsebino na "clicked"
$("li.entry").on("click", function( event ) {
  $(this).text("clicked");
});
```

Funkcija oziroma poslušalec, ki obravnava nek dogodek, prejme objekt `Event` [12]. Ta ima več lastnosti:

- `target` - element, ki je sprožil dogodek,
- `pageX` - položaj miške glede na levi rob dokumenta,
- `pageY` - položaj miško glede na zgornji rob dokumenta,
- `type` - vrsta dogodka (`click`, `mouseover`, `mousedown`, ...).

3.3.5 Delegacija dogodkov

Dogodki se propagirajo po DOM drevesu. To pomeni, da dogodek potuje navzgor po drevesu od začetnega elementa, nad katerim je bil sprožen. Kako poteka propagacija na preprostem HTML dokumentu je vidno na sliki 3.4.



Slika 3.4: Propagacija `click` dogodka.

Ko se sproži `click` dogodek na `<li>` elementu, dogodek potuje po prednikih `<li>`. Najprej po `<ul>` elementu, nato po `<body>` elementu, dokler dogodek ne doseže korena dokumenta.

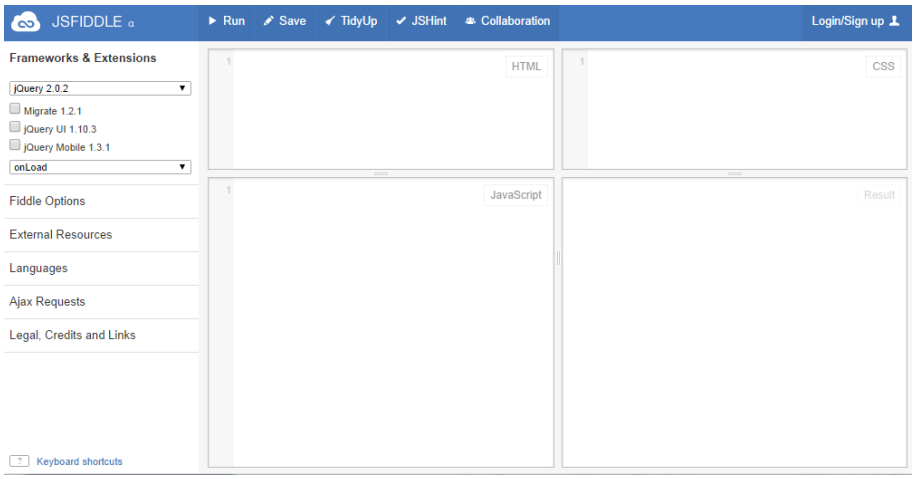
Zakaj je to pomembno? Če imamo v dokumentu skupino elementov in želimo, da se ob kliku na enega od elementov nekaj zgodi, ne prijavljamo poslušalca na vsak posamezen element, ampak ga prijavimo samo na predniku. Tak pristop delegacije dogodkov je uporaben predvsem v dveh primerih:

1. Imamo skupino z večjim številom elementov, na katere bi radi prijavili poslušalce. Namesto, da iteriramo čez posamezne elemente, poslušalca prijavimo samo enkrat na predniku.

2. Elemente v našem dokumentu ustvarjamo dinamično. Poslušalca ne moremo prijaviti vnaprej na neobstoječi element. Namesto, da vsakič, ko ustvarimo nov element, prijavljamo poslušalca na novo ustvarjeni element, to storimo samo enkrat na predniku.

3.4 JSFiddle

JSFiddle [20] je spletno orodje namenjeno hitri izdelavi prototipov. Na podlagi uporabniških izbir JSFiddle nudi okolje za hitro testiranje JavaScript, HTML in CSS kode. Orodje je vidno na sliki 3.5.



Slika 3.5: Orodje JSFiddle.

3.5 JSHint

JSHint [21] je orodje namenjeno odkrivanju morebitnih napak in težav v programski kodi. Orodje pregleda program napisan v JavaScriptu in prikaže poročilo o potencialnih hroščih in napakah v našem programu.

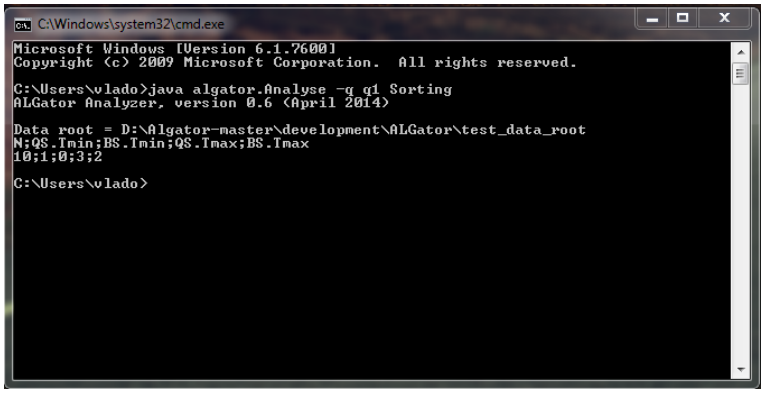
```
# namestitev
$ npm install jshint -g

# uporaba
$ jshint datoteka.js
```

JSHint se namesti s privzetim naborom opozoril, vendar ga je zelo preprosto prilagoditi lastnim potrebam.

3.6 ALGator

ALGator [2] je sistem za izvajanje algoritmov na podanih testnih podatkih ter analizo rezultatov izvajanja. Sistem omogoča dodajanje in upravljanje s poljubnim številom projektov. V okviru enega projekta je definiran problem, testne množice vhodnih podatkov ter način reševanja nalog tega problema. Projekt lahko vsebuje poljubno število algoritmov, ki naloge rešujejo na predpisan način. Sistem omogoča analizo izvajanja posameznega algoritma ter primerjavo med algoritmi istega projekta. Namen ALGator sistema je:

- reševanje definiranih problemov z različnimi algoritmi,
- analiza delovanja posameznega algoritma pri reševanju nalog danega problema,
- primerjava med različnimi algoritmi za reševanje istega problema.



Slika 3.6: Podsistem `Analizator`.

Sistem je sestavljen iz več podsistemov (`Konfigurator`, `Analizator`, ...), ki so medseboj povezani s konfiguracijskimi datotekami. V naši spletni aplikaciji uporabljamo predvsem podsistem `Analizator`, kateremu podamo podatkovno poizvedbo in ime projekta. `Analizator` pa nato izvede našo poizvedbo in izpiše podatke za podani projekt, kot je vidno na sliki 3.6.

3.7 C3.js

C3.js [9] je odprtokodna programska knjižnica namenjena risanju grafov. Temelji na knjižnici D3.js. Podpira izris več vrst grafov:

- črtni,
- stolpčni,
- ploščinski,
- tortni,
- kolobarni.

3.8 D3.js

D3.js [10] je JavaScript knjižnica za delo z dokumenti, ki temeljijo na podatkih. Knjižnica je izdana pod BSD licenco. Na primer, D3 lahko uporabimo za generiranje HTML tabele iz niza števil, ali pa uporabimo iste podatke, da ustvarimo interaktivni SVG črtni graf. To knjižnica stori na sledeči način:

1. podatke naloži v pomnilnik brskalnika,
2. podatke veže na elemente v dokumentu in po potrebi ustvari nove elemente,
3. z uporabo podatkov vezanih na elemente, elemente preoblikuje in jim nastavi oblikovne lastnosti.

3.8.1 Nalaganje podatkov

D3 je zmožen obdelave različnih vrst podatkov, tabela s podatki lahko vsebuje števila, nize in druge tabele. Prav tako lahko podatke naloži iz JSON zapisa, vgrajeno ima pa tudi metodo za nalaganje podatkov iz CSV datotek. Recimo, da imamo naše podatke v CSV zapisu shranjene v datoteki poimenovani `data.csv`. Z uporabo D3 lahko take podatke naložimo na sledeči način:

```
d3.csv("data.csv", function(data) {
        //tu nekaj storimo s podatki
});
```

Na voljo sta nam podobni metodi `d3.json()` in `d3.tsv()`, če želimo naložiti podatke v JSON ali TSV (Tab-separated values) zapisu:

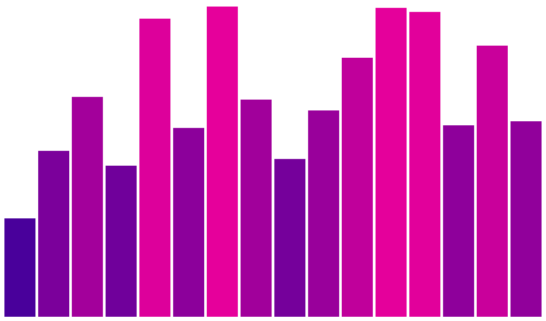
```
d3.json("data.json", function(data) {
        //tu nekaj storimo s podatki
});
```

Funkcija `d3.dsv()` nam omogoča nalaganje podatkov ločenih s poljubnim znakom. Recimo, da so naši podatki ločeni z znakom "|":

```
var dsv = d3.dsv("|", "text/plain");
dsv("data.csv", function(data) {
        //tu nekaj storimo s podatki
});
```

3.8.2 Risanje s podatki

Poglejmo si kako poteka risanje s pomočjo D3 in SVG. V našem primeru [4] bomo narisali zelo preprost stolpčni graf. Graf je prikazan na sliki 3.7.



Slika 3.7: Preprost stolpčni graf.

Stolpce smo narisali s pomočjo `<rect>` SVG elementa. Višino posameznega stolpca določa vrednost vnosa v tabeli s podatki. Naši podatki:

```
var podatki = [ 73, 123, 163, 112, 221, 140, 302, 161, 117,
                153, 192, 259, 226, 142, 201, 145];
```

Nato smo določili širino in višino svg elementa, padding med stolpci in njihovo širino:

```
var w = 500, h = 500, padding = 2, bar_width = 25;
var svg = d3.select("body")
    .append("svg")  //ustvarimo <svg>
    .attr("width", w)  //nastavimo sirino
    .attr("height", h);  //nastavimo visino
```

Na koncu še narišemo `<rect>` elemente. S pomočjo funkcije `attr()` nastavimo atribute `x`, `y`, `width`, `height` in `fill`, ki določajo koordinate, širino, višino in barvo stolpcev:

```
svg.selectAll("rect")
    .data(podatki)
    .enter()
    .append("rect")
    .attr({    //nastavimo atribute
        width: bar_width - padding,
        height: function(d) { return d; },
        x:  function(d, i) { return i * bar_width; },
        y:  function(d) {  return h - d; },
        fill: function(d) {  return "rgb(" + (d) + ", 0, 155)"; }
    });
```

Poglejmo si, kaj natanko se dogaja v zgornji kodi:

svg.selectAll(”rect”)

Izbere vse `<rect>` elemente. Ker ti še ne obstajajo, funkcija vrne prazen izbor elementov.

.data(podatki)

Razčleni naše podatke. V naši tabeli `podatki` je šestnajst vrednosti, vse kar sledi od te vrstice naprej, se izvede šestnajstkrat, enkrat za vsako vrednost.

.enter()

Funkcija `enter()` pogleda trenutno DOM izbiro in naše podatke. Če je podatkov več, kot je ustreznih DOM elementov, potem ustvari nov element. Nato poda referenco do novega elementa naslednjemu členu v verigi.

.append("rect")

V DOM pripne `<rect>` element.

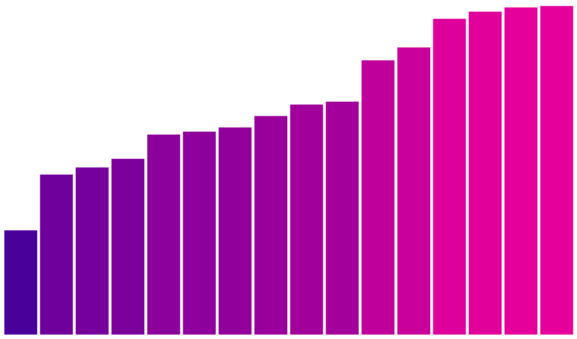
.attr()

Nastavi HTML atribut na elementu. V našem primeru smo nastavili atribute `x`, `y`, `width`, `height` in `fill`, ki določajo koordinate, širino, višino in barvo stolpcev.

Naš graf deluje, ampak je popolnoma statičen. Recimo, da želimo da se ob kliku na naš graf stolpci razvrstijo po velikosti, kot je vidno na sliki 3.8. Na `<svg>` element prijavimo poslušalca na sledeči način:

```
d3.select("svg").on("click", function () {
    svg.selectAll("rect")
        .sort(function (a, b) { return d3.ascending(a, b); })
        .transition()
        .duration(1000)
        .attr("x", function (d, i) { return i * bar_width; });
});
```

S pomočjo funkcije `sort()` razvrstimo elemente v grafu po velikosti, od najmanjšega do največjega. Funkcija `transition()` animira prehod stolpcev na nove koordinate, `duration()` pa določa trajanje prehoda v milisekundah.



Slika 3.8: Stolpci razvrščeni od najmanjšega do največjega.

Poglavje 4

Razvoj aplikacije

Namen naše aplikacije je grafičen in tabelaričen prikaz podatkov izbranega ALGator projekta. Vsak ALGator projekt lahko vsebuje poljubno število algoritmov, ki naloge rešujejo na predpisan način. Spletna aplikacija nam omogoča vizualizacijo in primerjavo rezultatov med različnimi algoritmi pri reševanju nekega problema. Aplikacijo sestavljata dve glavni komponenti:

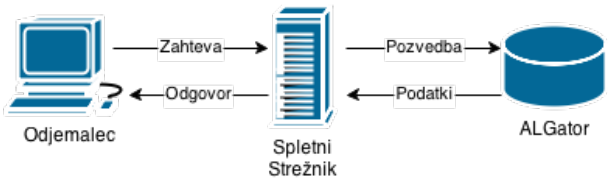
1. komponenta za sestavljanje poizvedb,
2. komponenta za risanje grafov.

4.1 Arhitektura aplikacije

Razvita spletna aplikacija uporablja tipično arhitekturo odjemalec/strežnik. Uporabnik aplikacije (odjemalec) zahteva določeni vir, nakar strežnik odgovori na zahtevo odjemalca. Procesiranje podatkov je izvedeno na strežniku, medtem se na odjemalčevi strani nahaja ustrezen uporabniški vmesnik. Arhitektura aplikacije je prikazana na sliki 4.1. Pri načrtovanju smo v obzir vzeli sledeče smernice:

- komponente aplikacije naj bodo zasnovane modularno. V praksi to pomeni, da se lahko posamezne komponente uporabi samostojno, neodvisno od ostalih modulov, komunikacija med komponentami pa poteka preko vnaprej določenih metod in vmesnikov,

- deli aplikacije naj bodo logično razdeljeni, kar bo v prihodnosti olajšalo vzdrževanje programske kode in dodajanje novih funkcij,
- aplikacije naj beleži napake v dnevniško datoteko. Dnevnik se potem lahko uporablja pri odkrivanju problemov v aplikaciji.



Slika 4.1: Arhitektura aplikacije.

Na odjemalčevi strani se za prikaz uporabniškega vmesnika in podatkov uporablja tehnologije HTML, CSS in SVG ter JavaScript programske knjižnice jQuery, D3.js in C3.js. Orodje Bootstrap je uporabljeno za razporeditev in izgled elementov grafičnega vmesnika. Podrobnejša predstavitev omenjenih orodij se nahaja v prejšnjih poglavjih.

Strežniški del sistema, ki obdeluje odjemalčeve zahteve, je napisan v programskem jeziku Python in spletnem ogrodju Flask. Sistem ALGator uporabljamo za izvajanje pozvedb. Odjemalec s pomočjo uporabniškega vmesnika izbere poizvedbo, nakar sistem ALGator izvede zahtevano podatkovno poizvedbo in nam vrne podatke za podani projekt.

4.2 Datotečna struktura

Deli aplikacije so razdeljeni v več mapah in podmapah, kot je vidno na sliki 4.2. V korenu se nahaja mapa `algatorweb`. V mapi `app` se nahaja paket s Python programsko kodo. V mapi `static` se hranijo statične datoteke, `templates` vsebuje HTML predloge. Sledi opis pomembnejših datotek:

algator.conf

Nastavitve v INI [17] formatu. V datoteki sta predvsem pomembni vrednosti `algator_data_root` in `algator_jar`.

algatorweb.py

Python programska koda.

algatorweb.wsgi

WSGI datoteka, potrebna za izvajanje aplikacije na strežnikih, ki podpirajo specifikacijo WSGI.

apache_sample_conf

Delna Apache konfiguracija, namenjena pomoči pri nameščanju aplikacije na strežnik Apache.

error.log

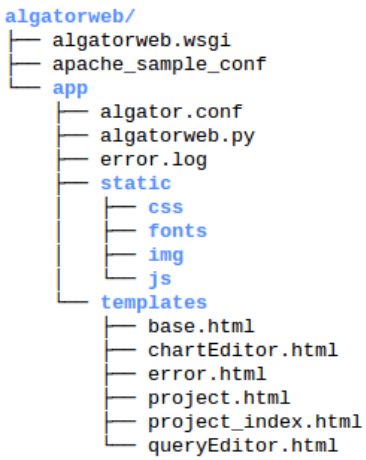
Dnevnik napak. V datoteko se beležijo napake in izjeme (ang. exceptions) do katerih lahko pride med izvajanjem.

algatorweb.js

Glavna JavaScript datoteka.

algatorweb.css

Glavna CSS datoteka.



Slika 4.2: Datotečna struktura aplikacije.

4.3 Komponenta za sestavljanje poizvedb

Komponenta za sestavljanje poizvedb opravlja več funkcij:

- prikaz in posodobitve uporabniškega vmesnika za sestavljanje poizvedb,
- v primeru spremembe poizvedbe, od strežnika zahteva nove podatke,
- prikaz in posodobitve trenutno izbrane poizvedbe,
- prikaz podatkov za trenutno izbrano poizvedbo v HTML tabeli,
- o spremembah podatkov, obvešča komponento za risanje grafov.

Programska koda komponente se nahaja v datoteki `algatorweb.js`, HTML predloga pa je shranjena v `queryEditor.html`. Za implementacijo smo uporabili JavaScript načrtovalski vzorec (ang. design pattern) "modul" [5]. V naši spletni aplikaciji inicializacija komponente poteka na sledeči način:

```
queryEditor.init(projectName);
var query = util.getParameterFromURL("query");
if (query !== "") {
        queryEditor.setQuery(query);
}
```

Komponento inicializiramo s pomočjo metode `init()`, kateri podamo niz z imenom projekta. Takoj zatem preverimo prisotnost parametra `query`. Če je parameter nastavljen, potem s pomočje metode `setQuery()` nastavimo trenutno poizvedbo. Podrobnejši opis metod komponente se nahaja v tabeli A.1.

4.4 Komponenta za risanje grafov

Namen komponente za risanje grafov je:

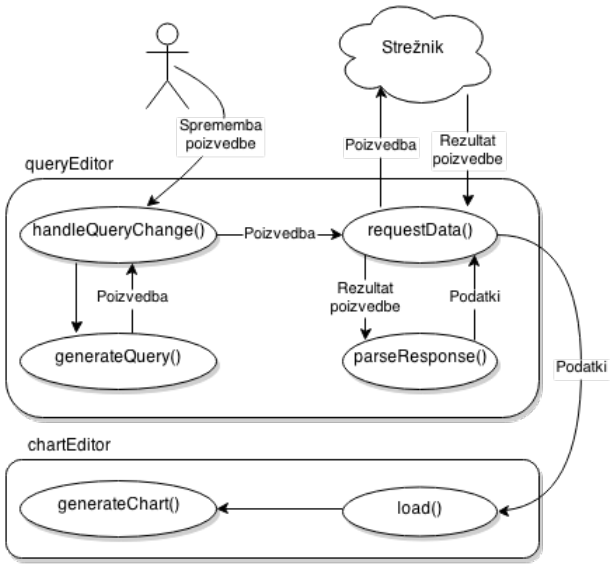
- prikaz in posodobitve uporabniškega vmesnika za nastavljanje grafa,
- prikaz in posodobitve samega grafa.

Programska koda komponente se nahaja v datoteki `algatorweb.js`, HTML predloga pa je shranjena v `chartEditor.html`. Za implementacijo smo, kot v primeru komponente za sestavljanje poizvedb, uporabili načrtovalski vzorec "modul" [5]. Na podoben način poteka tudi inicializacija komponente:

```
chartEditor.init("#chart");
```

Metodi `init()` podamo selektor, ki označuje `<div>` element, znotraj katerega se bo nahajal zahtevani graf. Prvotno komponenta za nas generira prazen graf. Novi podatki se nato naložijo s pomočjo metode `load()`. V razviti aplikaciji to počne komponenta za sestavljanje poizvedb, kadarkoli od ALGator sistema prejme nove podatke. V primeru, da je rezultat poizvedbe prazna tabela, se uporabi metoda `unload()`, katera ponastavi vrednosti grafa. Podrobnejši opis metod komponente se nahaja v tabeli A.2.

4.5 Interakcija med komponentama



Slika 4.3: Diagram poteka.

Interakcija med komponentama je prikazana na sliki 4.3. Ko pride do spremembe poizvedbe v komponenti `queryEditor`, se izvede `handleQueryChange()`.

Ta s pomočjo metode `generateQuery()` generira novo ALGator poizvedbo in pokliče metodo `requestData()`. Metoda `requestData()` nato poizvedbo pošlje spletnemu strežniku z uporabo tehnike Ajax, kjer poizvedbo izvede sistem ALGator. Rezultat se nato vrne odjemalcu v obliki teksta. Rezultat razčlenimo s pomočjo metode `parseResponse()`, ki podatke razčleni v dvodimenzionalno tabelo. Primer take tabele:

```
[["N", "QuickSort.Tmax", "QuickSort.Tmin"],
 ["10", "21", "3"],
 ["30", "7", "4"]
];
```

Podatki se nato podajo naprej komponenti `chartEditor`, specifično se v ta namen uporabi metoda `load()`. Nazadnje se posodobi legenda grafa in iz podanih podatkov izriše graf.

4.6 Prilagajanje komponent

V tem poglavju bomo pokazali, kako lahko posamezno komponento prilagodimo lastnim potrebam. V našem primeru bomo opisali korake potrebne, da komponenti za sestavljanje poizvedb dodamo funkcionalnost shranjevanja podatkov ter poizvedb v tekstovne datoteke, kot je prikazano na sliki 4.4.

Slika 4.4: Shranjevanje poizvedb in podatkov.

V primeru klika na črno ikono za shranjevanje, se bo v datoteko zapisala poizvedba oziroma podatki poizvedbe. Poizvedbo bomo shranili v JSON formatu, medtem ko bomo podatke shranili v CSV formatu. Kot prvi korak, odpremo HTML predlogo komponente za sestavljanje poizvedb (`queryEditor.html`) in dodamo dve `<img>` znački:

```
<img class="clickable download-query"
        src="{{ url_for('static', filename='img/save.png') }}">
```

```
...
<img class="clickable download-data"
        src="{{ url_for('static', filename='img/save.png') }}">
```

Značkama `<img>` smo dodelili razreda `download-query` in `download-data`, ki ju bomo potrebovali kasneje, ko bomo prijavljali poslušalca na dodana elementa. Razred `clickable` uporabimo izključno za nastavljanje CSS oblikovnih lastnosti dodanih `<img>` značk. Želimo, da se oblika miškinega kazalca spremeni, ko se kazalec premakne čez dodani ikoni. To storimo tako, da v datoteko `algatorweb.css` dodamo sledeči zapis:

```
img.clickable {
        cursor: pointer;
}
```

Naš naslednji korak je prijava poslušalcev na dodanih ikonah. Odpremo datoteko `algatorweb.js` in v komponento `queryEditor` dodamo sledeči konstanti:

```
var DOWNLOAD_QUERY = "img.download-query";
var DOWNLOAD_DATA = "img.download-data";
```

V isti datoteki se premaknemo na metodo `wireEvents()` (tabela A.1), ki skrbi za nastavljanje poslušalcev v komponenti . Kadarkoli bo prišlo do klika na posamezni ikoni, bomo klicali metodo `handleQueryDownload()` ali `handleDataDownload()`. Na koncu metode `wireEvents()` dodamo naslednji vrstici:

```
$(DOWNLOAD_QUERY).on("click", handleQueryDownload);
$(DOWNLOAD_DATA).on("click", handleDataDownload);
```

Zaradi razlik pri implementaciji programskih vmesnikov v spletnih brskalnikih, generiranje in shranjevanje datotek na strani odjemalca poteka na različne načine. Za ta namen bomo uporabili programsko knjižnico FileSaver.js [14], ki poskrbi za podrobnosti implementacije na posameznih spletnih brskalnikih. Potrebna je samo še implementacija metod `handleQueryDownload()` in `handleDataDownload()`, kjer se bo nahajala logika za shranjevanje poizvedb in podatkov. Začnimo kar z metodo `handleQueryDownload()`:

```
function handleQueryDownload() {
    if (query !== undefined) {
        var queryText = JSON.stringify(query, undefined, 2);
        var blob = new Blob([queryText],
            {type: "text/plain;charset=utf-8"});
        saveAs(blob, project + "-query.json");
    }
}
```

Če je spremenljiva `query` nastavljena, potem v spremenljivko `queryText` shranimo niz, ki predstavlja našo poizvedbo. Nato kličemo metodo knjižnice FileSaver.js `saveAs()`, ki poizvedbo shrani v JSON datoteko. Manjka samo še metoda `handleDataDownload()`:

```
function handleDataDownload() {
    if (data !== undefined) {
        var csvText = data.map(function(line) {
            return line.join(",");
        }).join("\n");

        var blob = new Blob([csvText], {type: "text/plain;charset=utf-8"});
        saveAs(blob, project + "-data.csv");
    }
}
```

Spremenljivka `data` kaže na dvodimenzionalno tabelo v kateri so shranjeni naši podatki. Primer tabele s podatki:

```
[["a","b","c"],
 [1,2,3]];
```

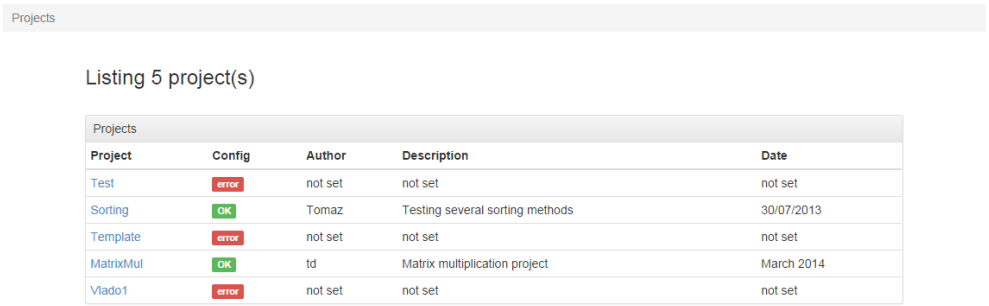
Vsebino tabele shranimo v niz v CSV formatu:

```
"a,b,c
1,2,3"
```

Za zaključek niz samo še shranimo v CSV datoteko z metodo `saveAs()`.

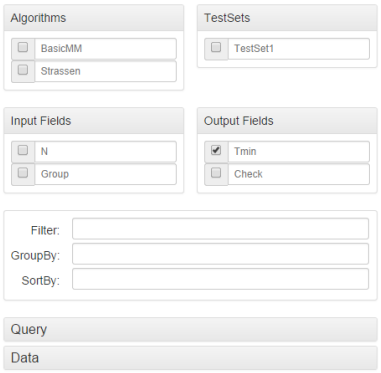
4.7 Uporaba aplikacije

Uporabniku aplikacije se pri začetnem dostopu s spletnim brskalnikom, prikaže seznam ALGator projektov, ki so na voljo. Projekti, ki jim manjka konfiguracijska datoteka, so v tabeli označeni z rdečo oznako. Poleg imena projekta so v tabeli navedeni tudi avtor, opis projekta in datum zadnje spremembe projekta. Seznam projektov je prikazan na sliki 4.5.



Slika 4.5: Seznam ALGator projektov.

Po izbiri projekta nas aplikacija preusmeri na projektno stran. Na skrajni levi strani se nahaja vmesnik za sestavljanje poizvedb, na sredini je prikazan graf, na desni strani pa se nahaja vmesnik za prikaz grafa.



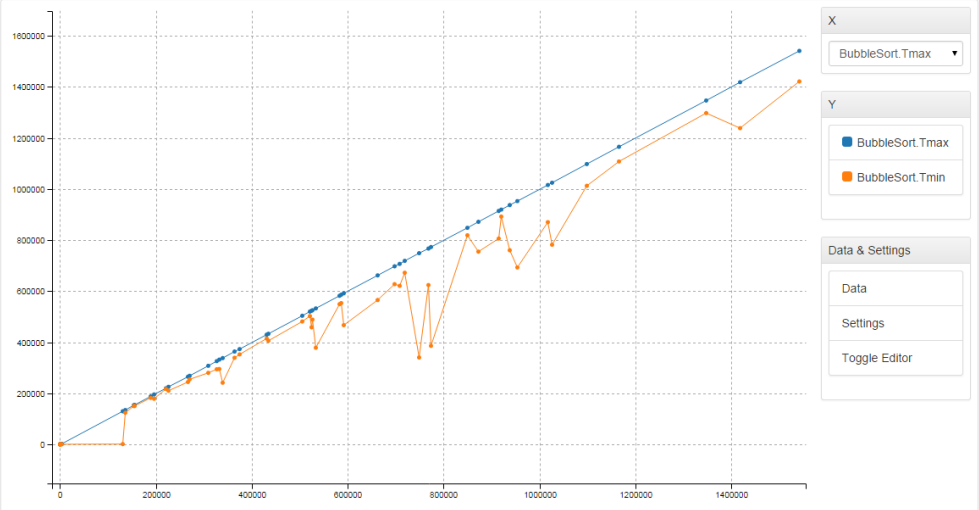
Slika 4.6: Vmesnik za sestavljanje poizvedb.

Vmesnik za sestavljanje poizvedb, kot je prikazan na sliki 4.6, je razdeljen na več odsekov. V zgornjem levem delu se nahajajo odseki za izbiro algoritmov,

testnih množic ter vhodnih in izhodnih parametrov. Ti določajo obseg naše poizvedbe, na primer, če želimo določen algoritem vključen v rezultat poizvedbe, to označimo na ustreznem potrditvenem polju. Na podlagi naše izbire se nato samodejno generira nova poizvedba. Kako je videti naša trenutna poizvedba si lahko ogledamo v odseku "Query". Rezultat naše poizvedbe je prikazan v odseku "Data", kjer se nahaja HTML tabela z zahtevanimi podatki.

Ob vsaki spremembi rezultatov poizvedbe, se podatki samodejno izrišejo na grafu, ki se nahaja na sredini strani. Kot je prikazano na sliki 4.7, se na skrajni desni strani nahajajo polja za nastavljanje osi in legenda grafa, katera ima predvsem sledeča namena:

- prikaz barv dodeljenih posameznim stolpcem,
- skrivanje posameznih stolpcev na grafu.



Slika 4.7: Prikaz grafa.

Pod legendo se nahaja več gumbov. S klikom na gumb "Data" se nam prikaže modalni dialog, ki vsebuje tekstovno polje s katerim lahko urejamo podatke. Nižje se nahaja gumb "Settings", s pomočjo katerega urejamo nastavitve grafa. Nastavljamo lahko lastnosti mreže in vrsto grafa (npr. tortni graf, kot je prikazan

na sliki 4.8). Na koncu se nahaja še gumb "Toggle Editor", ki skrije vmesnik za sestavljanje komponent in ponovno izriše graf čez celotno širino okna.

Preko parametrov naslovne vrstice lahko spletni aplikaciji podamo več argumentov:

query

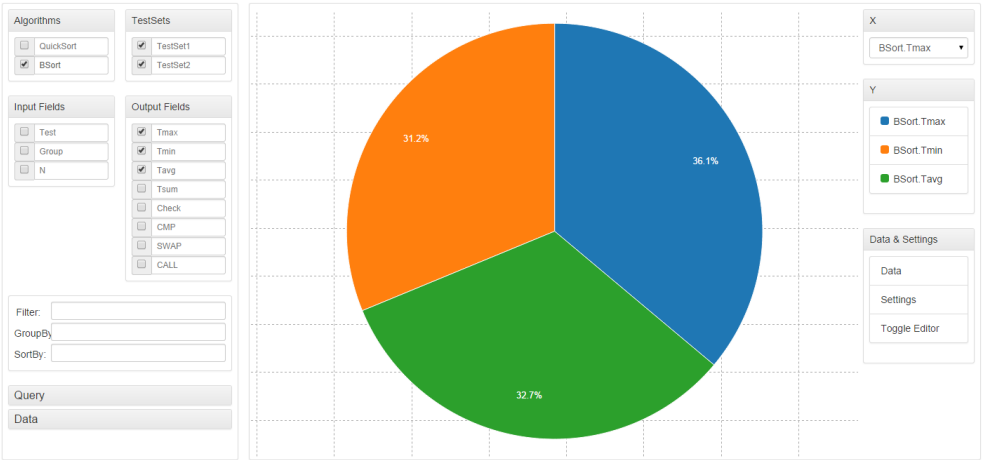
Vrednost trenutne poizvedbe. Če je parameter prisoten, ko se naloži projektna stran, se samodejno izvede poizvedba podana v parametru in izriše zahtevani graf. Parameter se prav tako samodejno nastavi ob vsaki spremembi poizvedbe.

hideEditor

Prisotnost parametra (`hideEditor=true`) nakazuje, da bo vmesnik za sestavljanje poizvedb privzeto skrit. Namesto njega bo čez celotno širino okna izrisan graf.

dbg

Parameter namenjen razhroščevanju (`dbg=true`). V primeru težav ga lahko uporabimo za podrobnejši izpis napake.



Slika 4.8: Prikaz tortnega grafa.

Poglavje 5

Nastavitve in namestitev

5.1 Zahteve

Za delovanje spletne aplikacije potrebujemo:

1. Python,
2. Flask ogrodje [15],
3. ALGator [2],
4. Spletni strežnik, ki podpira WSGI (npr. Apache in mod_wsgi).

5.2 Nastavitve

Nastavitve aplikacije so shranjene v datoteki `algator.conf`, ki se nahaja v `algatorweb/app` mapi. V datoteki moramo pred uporabo nastaviti dve vrednosti:

algator_data_root
: Pot do mape z Algator podatki.

algator_jar
: Pot do programa ALGator.jar.

Recimo, da se datoteka ALGator.jar nahaja v `/home/user/ALGator/dist`, Algator podatki pa se nahajajo v mapi `/home/user/ALGator/test_data_root`. Vrednosti v datoteki `algator.conf` nastavimo na sledeči način:

```
algator_data_root = /home/user/ALGator/test_data_root
algator_jar = /home/user/ALGator/dist/ALGator.jar
```

Preden se lotimo nameščanja aplikacije na strežnik Apache je pametno, da nove nastavitve preizkusimo s pomočjo strežnika vgrajenega v Flask ogrodje. V ukazni vrstici zaženemo:

```
$ python algatorweb.py
 * Running on http://0.0.0.0:5000/
```

Delovanje aplikacije nato preizkusimo na http://127.0.0.1:5000 s spletnim brskalnikom.

5.3 Namestitev

Poglejmo si še kako poteka namestitev spletne aplikacije na strežnik Apache in mod_wsgi. Aplikacijo lahko namestimo na poljubno lokacijo, v našem primeru smo jo namestili v mapo `/var/www`, kot je prikazano na sliki 5.1.

```
/var/www/algatorweb
├── algatorweb.wsgi
├── apache_sample_conf
└── app
    ├── algator.conf
    ├── algatorweb.py
    ├── error.log
    ├── static
    └── templates
```

Slika 5.1: Namestitev v /var/www.

5.3.1 Datoteka `algatorweb.wsgi`

Če želimo našo aplikacijo uporabljati na mod_wsgi potrebujemo `algatorweb.wsgi` datoteko. Ta je vključena v `algatorweb` direktoriju. Datoteko odpremo in uredimo spremenljivko `path` tako, da kaže na mapo z datoteko `algatorweb.py`.

```
path = '/var/www/algatorweb/app'
```

5.3.2 Konfiguracija strežnika Apache

Sledi konfiguracija Apache strežnika. Vzorčna konfiguracija se nahaja v datoteki `apache_sample_conf`. Za nas so pomembne predvsem direktive:

WSGIScriptAlias

Pot do wsgi datoteke.

Directory

Pot do mape z aplikacijo.

WSGIDaemonProcess

Nastavitve procesa pod katerim bo delovala aplikacija. Če sta nastavljena parametra `user` in `group` bo proces tekel pod uporabnikom in skupino določenima s parametroma, v nasprotnem primeru bo proces tekel pod privzetim Apache uporabnikom.

```
<VirtualHost *:80>
        ServerName mywebsite.com
        ServerAdmin admin@mywebsite.com

        WSGIDaemonProcess algatorweb
        # WSGIDaemonProcess algatorweb user=www-data group=www-data threads=5

        WSGIScriptAlias / /var/www/algatorweb/algatorweb.wsgi

        <Directory /var/www/algatorweb>
                WSGIProcessGroup algatorweb
                WSGIScriptReloading On
                Order deny,allow
                Allow from all
        </Directory>
</VirtualHost>
```

Ko zaključimo z urejanjem Apache nastavitev, strežniku povemo naj ponovno naloži konfiguracijsko datoteko:

```
$ sudo /etc/init.d/apache2 reload        # ali
$ sudo /sbin/service httpd reload
```

5.3.3 Odpravljanje težav

V primeru, da aplikacija ne deluje, čeprav smo opravili zgornje korake, nam je v pomoč lahko Apache dnevnik napak, čigar ime in lokacija sta določena z direktivo `ErrorLog` v Apache nastavitvah. V dnevnik napak se zabeležijo napake, do katerih pride pri obdelavi http zahtev. Tipično se nahaja na:

Debian, Ubuntu

`/var/log/apache2/error.log`

Fedora Core, CentOS

`/var/log/httpd/error_log`

openSUSE, SLES

`/var/log/apache2/error_log`

Poglavje 6

Sklepne ugotovitve

V diplomskem delu smo želeli pregledati področje vizualizacije podatkov s pomočjo programskega jezika JavaScript. Ogledali smo si delovanje modernih tehnologij in programskih knjižnic iz tega področja in nato z uporabo predstavljenih tehnologij razvili spletno aplikacijo za izvajanje poizvedb in vizualizacijo rezultatov sistema ALGator. Razvita aplikacija je modularna, tako da jo bo mogoče v prihodnosti enostavno nadgrajevati. Veliko časa je bilo namenjega razvoju samega uporabniškega vmesnika, kateri je šel skozi več različnih iteracij, dokler nismo z njim bili končno zadovoljni.

Ena od večjih prednosti spletnih aplikacij pred tradicionalnimi namiznimi aplikacijami je predvsem v tem, da ni potreben korak nameščanja aplikacije. Enako velja tudi za našo razvito rešitev. Namesto nameščanja na posamezne računalnike, aplikacijo na strežnik namesti skrbnik sistema, uporabniki pa nato za dostop do aplikacije potrebujejo samo spletni brskalnik. Tak pristop omogoča tudi poenostavljeno vzdrževanje in posodabljanje aplikacije.

Kljub temu, da je razvita spletna aplikacija že primerna za uporabo, so v prihodnosti mogoče številne manjše in večje izboljšave. Tekom razvoja aplikacije se je porodila precej ambiciozna ideja, strežniški del aplikacije bi namesto v okolju Python, implementirali v Java razvojnem okolju, kar bi omogočalo boljšo integracijo s samim sistemom ALGator. Taka rešitev bi nato lahko neposredno uporabljala ALGator API, kar bi pri uvajanju novih funkcij pomenilo manj podvojene izvorne kode in s tem tudi manj hroščev.

Literatura

[1] B. Cooper (2010), The Gradual Disappearance Of Flash Websites, dostopno na:
http://www.smashingmagazine.com/2010/04/12/the-gradual-disappearance-of-flash-websites/

[2] T. Dobravec (2014), ALGator, dostopno na:
https://github.com/ALGatorDevel/Algator

[3] M. Grinberg, Flask Web Development, Developing Web Applications with Python, O'Reilly Media, 2014.

[4] S. Murray, Interactive Data Visualization for the Web, O'Reilly Media, str. 7–14, 49–57, 63–107, 2013.

[5] A. Osmani, Learning JavaScript Design Patterns, O'Reilly Media, 2014.

[6] (2014) AJAX, dostopno na:
https://developer.mozilla.org/en/docs/AJAX

[7] (2014) Bootstrap (front-end framework), dostopno na:
http://en.wikipedia.org/wiki/Bootstrap_(front-end_framework)

[8] (2014) Bootstrap CSS, dostopno na:
http://getbootstrap.com/css/

[9] (2014) C3.js, dostopno na:
http://c3js.org/

[10] (2014) D3.js, dostopno na:
http://d3js.org/

[11] (2014) Document Object Model (DOM), dostopno na:
https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

[12] (2014) Event Object - jQuery API Documentation, dostopno na:
http://api.jquery.com/category/events/event-object/

[13] (2014) Event reference, dostopno na:
https://developer.mozilla.org/en-US/docs/Web/Events

[14] (2014) FileSaver.js, dostopno na:
https://github.com/eligrey/FileSaver.js/

[15] (2014) Flask (A Python Microframework), dostopno na:
http://flask.pocoo.org/

[16] (2014) HTML5, dostopno na:
https://developer.mozilla.org/en/docs/web/Guide/HTML/HTML5

[17] (2014) INI file, dostopno na:
http://en.wikipedia.org/wiki/INI_file

[18] (2014) JavaScript, dostopno na:
https://developer.mozilla.org/en/docs/Web/JavaScript

[19] (2014) jQuery Basics, dostopno na:
http://jqfundamentals.com/chapter/jquery-basics

[20] (2014) JSFiddle, dostopno na:
http://jsfiddle.net/

[21] (2014) JSHint, dostopno na:
http://www.jshint.com/

[22] (2014) Paths, dostopno na:
http://www.w3.org/TR/SVG/paths.html

[23] (2014) SVG, dostopno na:
https://developer.mozilla.org/en-US/docs/Web/SVG

Dodatek A

Opis metod

Metoda	Dostop	Opis
`init()`	javen	Poskrbi za inicializacijo komponente. Kot argument podamo niz, ki vsebuje ime projekta. Ime projekta je potrebno za izvajanje poizvedbe v ALGator sistemu.
`getQuery()`	javen	Vrne niz s trenutno vrednostjo izbrane poizvedbe.
`setQuery()`	javen	Nastavi novo vrednost poizvedbe in poskrbi, da se posodobijo vrednosti elementov grafičnega vmesnika.
`getData()`	javen	Vrne rezultat zadnje poizvedbe. Podatki se vrnejo v dvodimenzionalni tabeli.
`generateQuery()`	privaten	Generira novo poizvedbo iz trenutne uporabnikove izbire.
`parseResponse()`	privaten	Razčleni ALGatorjev odgovor na zahtevano poizvedbo. Razčlenjene podatke vrne v dvodimenzionalni tabeli.
`requestData()`	privaten	Izvede AJAX klic, ki zahteva nove podatke od ALGator sistema. Kot argument metodi podamo ALGator poizvedbo.
`handleQueryChange()`	privaten	Poslušalec za dogodke v komponenti za sestavljanje poizvedb.
`wireEvents()`	privaten	Nastavi poslušalce za dogodke v komponenti. Metoda se kliče v postopku inicializacije komponente.
`populateTable()`	privaten	Generira in prikaže novo HTML tabelo z ALGator podatki.

Tabela A.1: Metode komponente za sestavljanje poizvedb (`queryEditor`).

Metoda	Dostop	Opis
`init()`	javen	Poskrbi za inicializacijo komponente. Kot argument metodi podamo selektor za `<div>` element v katerem želimo, da se nahaja graf.
`load()`	javen	Naloži nove podatke. Kot argument metodi podamo dvodimenzionalno tabelo s podatki.
`reload()`	javen	Ponovno izriše trenutni graf in generira seznam elementov za izbiro osi grafa.
`toggleQueryEditor()`	javen	Nastavlja vidljivost komponente za sestavljanje poizvedb.
`wireEvents()`	privaten	Poskrbi za prijavo poslušalcev za dogodke v komponenti. Metoda se kliče v postopku inicializacije komponente.
`populateXPanel()`	privaten	Generira in prikaže seznam elementov za izbiro x osi.
`populateYPanel()`	privaten	Generira in prikaže seznam elementov za izbiro y osi.
`generateXColumn()`	privaten	Generira dodaten stolpec v tabeli s podatki za prikaz x osi.
`generateChart()`	privaten	Iz podanih podatkov nariše graf. Kot argument metodi podamo podatke shranjene v stolpcih v dvodimenzionalni tabeli.

Tabela A.2: Metode komponente za risanje grafov (`chartEditor`).